PDR.01 SDP Architecture

Document number………………………………………………………………………………………. SKA-TEL-SDP-0000013

Context…….. ARCH

Revision………..2

Author…………………………………………………………………………………………. ……………………………Nikolic

Release Date……2015-02-09

Document Classification………………………………………………………………………………………Unrestricted

Status………………………………………………………………………………………. ……………………………………Draft

Name	Designation	Affiliation
Bojan Nikolic	Project Engineer	University of Cambridge
Signature & Date:	**Signature:** [illegible] Bojan Nikolic (Feb 9, 2015) **Email:** b.nikolic@mrao.cam.ac.uk	

Name	Designation	Affiliation
Paul Alexander	SDP Lead	University of Cambridge
Signature & Date:	**Signature:** Paul Alexander Paul Alexander (Feb 9, 2015) **Email:** pa@mrao.cam.ac.uk	

Version	**Date of Issue**	**Prepared by**	**Comments**
2	2015-02-09	B. Nikolic, R. Simmonds, C. Broekema, R. Nijboer, A. Wicenec, M. Dolensky	

ORGANISATION DETAILS

Name	Science Data Processor Consortium

1 TABLE OF CONTENTS

2 LIST OF FIGURES

3 LIST OF TABLES

4 INTRODUCTION

This document provides the top-level design report for the SKA Science Data Processor (SDP) element for the preliminary design review. It describes the context and functions of the SDP within the SKA observatory, lists some of the principles of the SDP design and the key top-level design decisions. Also described is the current decomposition of SDP into subsystems (which are described in their respective initial subsystems reports in the preliminary design review).

5 REFERENCES

5.1 APPLICABLE DOCUMENTS

The following documents are applicable to the extent stated herein. In the event of conflict between the contents of the applicable documents and this document, **the applicable documents** shall take precedence.

Reference Number	**Reference**
AD1	SKA1 SYSTEM BASELINE DESIGN, SKA-TEL-SKO-DD-001
AD2	MISCELLANEOUS CORRECTIONS TO THE BASELINE DESIGN
AD3	SKA PHASE 1 SYSTEM (LEVEL 1) REQUIREMENTS SPECIFICATION
AD4	SKA1 INTERFACE CONTROL DOCUMENT SDP TO CSP, SKA-TEL.SDP.SE-TEL.CSP.SE-ICD-001, Revision E
AD5	SKA1 INTERFACE CONTROL DOCUMENT SDP TO TM, SKA-TEL.SDP.SE-TEL.TM.SE-ICD-001 Revision A
AD6	INTERFACE CONTROL DOCUMENT SADT TO SDP, SKA-TEL.SADT.SE-TEL.SDP.SE-ICD-001, Revision H
AD7	SKA1 INTERFACE CONTROL DOCUMENT SDP TO INFRA-AUS AND INFRA-SA, Revision B, dated 2015-01-30

5.3 REFERENCE DOCUMENTS

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

Reference Number	**Reference**
RD1	Designing Cloud and Grid Computing Systems with InfiniBand and High-Speed Ethernet, Dhabaleswar K. (DK) Panda, Sayantan Sur, Tutorial at CCGrid '11
RD2	Managing Skew in Hadoop, YongChul Kwon, Kai Ren, Magdalena Balazinska , and Bill Howe, IEEE DATA ENG. BULL., VOL
RD3	MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
RD4	Data parallel algorithms, W. Daniel Hillis and Guy L. Steele, Jr, Commun. ACM 29, 12 (December 1986), 1170-1183. DOI=10.1145/7902.7903 http://doi.acm.org/10.1145/7902.7903
RD5	Parametric Models of SDP Compute Requirements, SKA-TEL-SDP-0000003
RD6	NVIDIA Tesla: A Unified Graphics and Computing Architecture, Lindholm, E.; Nickolls, J.; Oberman, S.; Montrym, J., Micro, IEEE , vol.28, no.2, pp.39,55, March-April 2008
RD7	Multi-frequency synthesis techniques in radio interferometric imaging., Sault, R. J.; Wieringa, M. H., Astronomy and Astrophysics Suppl. 108, 585-594 (1994) (A&AS Homepage)
RD8	Apache Hadoop 2.5.1 Online Documentation, Apache Hadoop Project Team, http://hadoop.apache.org/docs/current/
RD9	CASA User Reference & Cookbook, Editor: J urgen Ott – Project Scientist, Jeff Kern – CASA Project Manager, NRAO Technical Manual, http://casa.nrao.edu/Doc/Cookbook/casa_cookbook.pdf

RD10	Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines, Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe, *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation (PLDI '13). ACM, New York, NY, USA, 519-530*
RD11	Advances in dataflow programming languages, Wesley M. Johnston, J. R. Paul Hanna, and Richard J. Millar, *ACM Comput. Surv. 36, 1 (March 2004), 1-34.*
RD12	DAGuE: A generic distributed DAG Engine for High Performance Computing, Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemarinier, P., Dongarra, J., Parallel Computing
RD13	Swift: Fast, reliable, loosely coupled parallel computation, Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde, In Proc. Workshop on Scientific Workflows, 2007
RD14	Turbine: A distributed-memory dataflow engine for high performance many-task applications, Justin M. Wozniak, Timothy G. Armstrong, Ketan Maheshwari, Ewing L. Lusk, Daniel S. Katz, Michael Wilde, Ian T. Foster, http://swift-lang.org/papers/pdfs/Turbine_2013.pdf
RD15	A survey of power and energy efficient techniques for high performance numerical linear algebra operations, Li Tan, Shashank Kothapalli, Longxiang Chen, Omar Hussaini, Ryan Bissiri, Zizhong Chen, Parallel Computing, www.cs.ucr.edu/~ltan003/Publications/PARCO'14.pdf
RD16	SKA Software & Computing CoDR Documentation Set, SKA Organisation & The Authors, https://indico.skatelescope.org/conferenceDisplay.py?confId=176
RD17	SKA SDP PRELIMINARY PLAN FOR CONSTRUCTION, SDP Consortium
RD18	Exascale Computing Trends: Adjusting to the New Normal for Computer Architecture, Kogge, P.; Shalf, J.,, *Computing in Science & Engineering , vol.15, no.6, pp.16,26, Nov.-Dec. 2013,* http://dx.doi.org/10.1109/MCSE.2013.95
RD19	Asynchrony in parallel computing: From dataflow to multithreading, Jurij Silc and Borut Robic and Theo Ungerer, Journal of Parallel and Distributed Computing Practices, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.128.1374

RD20	Characterization of the Cray Aries Network, Brian Austin, NUG 2014, February 6, 2014, PDF of slides available on internet , ***https://www.nersc.gov/assets/pubs_presos/NUG2014Aries.pdf***
RD21	PDR.07 COSTING BASIS OF ESTIMATE, SKA-TEL-SDP-0000009, Ferdl Graser,
RD22	MPI: A Message-Passing Interface Standard Version 3.0, Message Passing Interface Forum, www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf,
RD23	A Large-Scale Study of Failures in High-Performance Computing Systems, Schroeder, B.; Gibson, G.A., *Dependable and Secure Computing, IEEE Transactions on , vol.7, no.4, pp.337,350, Oct.-Dec. 2010,* doi: 10.1109/TDSC.2009.4
RD24	A fault tolerant implementation of Multi-Level Monte Carlo methods, A fault tolerant implementation of Multi-Level Monte Carlo methods, Technical report, ETH, Department of Computer Science , 2103, http://dx.doi.org/10.3929/ethz-a-009922757
RD 25	SDP COMP element design report, PDR
RD 26	SDP Data element design report, PDR
RD 27	SDP Pipelines element design report, PDR
RD 28	SDP Data Delivery Platform element design report, PDR
RD 29	SDP Local Monitor and Control element design report, PDR
RD 30	SDP Dataflow Environment

6 Science Data Processor Element Context

The Science Data Processor (SDP) element of the SKA is responsible for the processing of SKA data output from the Central Signal Processing (CSP) element into science-ready data products, the safe keeping of these data products and the delivery of these products to external entities: users and regional science centres. The SDP is additionally responsible for computing and feeding back to the system some calibration solutions[1].

The observed data is ingested into the SDP from the Central Signal Processor (CSP) element while the metadata (describing the telescope configuration, observation being done, etc.) is ingested from the Telescope Manager (TM) element. The data transfer is via the Signal and Data Transport (SaDT) element. The SDP is controlled and monitored by the TM like all other elements of the SKA.

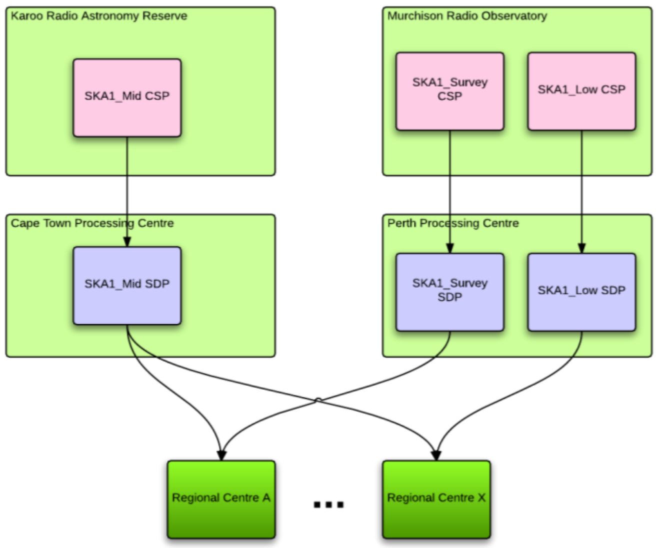

Figure 1: Physical deployment and context of the SDP element

One instance of the SDP element exists for each of the three telescopes of the SKA1. The SKA baseline design [AD01] seeks a single architecture for all three Science Data Processor instances, although the detailed hardware implementations may differ. We follow this approach and present a single architecture for all three instances. The differences in the instances shall be restricted primarily to the

[1] Full definition of the calibrations to be calculated in the SDP awaits the release of the SKA Calibration Strategy document.

unit numbers, the networking changes required to accommodate these and software configuration differences. It is also possible that some adjustment of working memory with compute nodes may be applied to each of the instances. Nevertheless their architecture is the same and no distinction between the instances will be made in this document. All instances of the SDP communicate with all regional centres.

6.1 Summary of SDP Functions

The top-level functions of the SDP can be divided into the following broad categories:

1. Ingest of data from the Central Signal Processor & Telescope Manager
2. Processing of input data into science-ready data products
3. Archiving the science data products permanently
4. Providing access to the science data so archived
5. Control, monitoring and feedback information back to the telescope as a system through the Telescope Manager

6.1.1 Ingest of data

The SDP shall be capable of storing input data for later (more complete) processing (F.3). For example, data for a deep continuum observation may be stored until the whole observation is complete and then fully processed while the next observation is being ingested. This means that SDP exposes a state which is longer-lived than a single observation which is the set of previous observations being held in the buffer. The TM will make decisions on when to process these.

The ingest function will also apply RFI flagging and possibly masking & excision (SKA1-SYS_REQ-2474, SKA1-SYS_REQ-2473, SKA1-SYS_REQ-2472) together with other possible functions which prepare the data for the processing to science products (e.g. merging of metadata, averaging, re-packaging and phase rotation)

6.1.2 Processing to science-ready data products

The functions here can be summarised as follows:

1. Processing of visibilities (i.e., the outputs of the correlator) into image cubes, or more precisely into image-spectral-polarisation datasets. A distinction is made between `continuum' processing (F.1, SKA1-SYS_REQ-2339) in which a minimal amount of spectral information is retained and 'spectral-line' mode (F.2, SKA1-SYS_REQ-2341, SKA1-SYS_REQ-2343) in which a large amount of information in the spectral dimension is retained. This distinction is necessary because of the large differences in output data volume, the computational savings that can be made by reducing in spectral dimension at intermediate stages during continuum processing and because the intrinsic noise due to the background radiation and the receiver (`thermal') noise is significantly reduced in the continuum dataset and therefore there is the possibility of higher dynamic range if effects other than thermal noise do not dominate. An additional imaging function for processing of drift scan interferometry (F.5) is proposed in anticipation of forthcoming Level 1 requirements.

2. Processing of the visibilities to identify time-varying sources in the sky at relatively high cadence and low latency (F.7, SKA1-SYS_REQ-2345). This is variously called `Slow Transients', `Fast Imaging', or `Transient Detection Pipeline'.
3. Processing of time series data to confirm and then time pulsars. The confirmation (F.10, SKA1-SYS_REQ-2129) stage processes relatively short amounts of data to ensure that pulsar candidates are true pulsars while the purpose of timing (F.9, SKA1-SYS_REQ-2130) is to extract accurate time-of-arrival information of pulses from known pulsars (whether found by the SKA or otherwise).
4. Processing of time series to detect very fast cadence transients (F.8, SKA1-SYS_REQ-2131) including the generation of alerts to be communicated via the TM interface.
5. The science analysis function (F.6), which consists of source finding and source stacking (SKA1-SYS_REQ-2335). In addition the SDP will support epoch of reionisation experiments by the function to store calibrated, source-subtracted and further averaged visibilities in the archive for further scientific analysis directly on the visibility data.

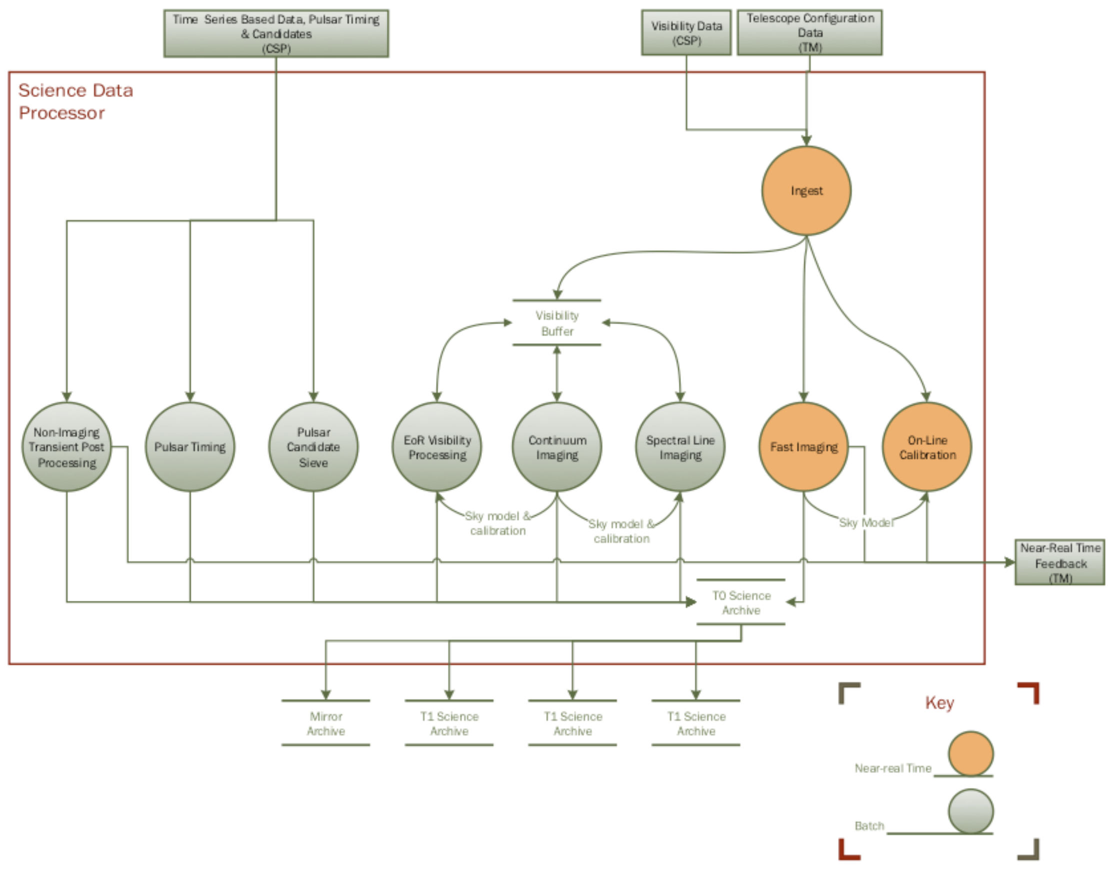

Figure 2: Summary of functions to produce science-ready data products, together with some ancillary functions and a schematic illustration of the key parts of the data flow between them.

6.1.3 Archiving of Science-Ready Data Products

This function encapsulates all of the actions required to accumulate the science-ready data products as described in the previous section so that they are permanently available (F.12, SKA1-SYS_REQ-2821). This archive is the only long-term store of the information observed by the telescope and by far the major[2] source of information for astronomers working with the SKA. Only the science data products produced by the SDP will be archived in the SDP archive, i.e., there will be no mechanism for elements other than to SDP to insert data in the archive (SKA1-SYS_REQ-2358).

6.1.4 Access to Science-Ready Data

The SDP element will also provide access to all of the data in the Science Ready Archive. Access will be provided directly both to users (i.e., astronomers) and also to institutes that may wish to obtain significant fraction of the archived data in order to provide some SKA data access facilities locally to astronomers. The balance between these functions is still to be determined and will be informed largely by the SKA operations plans and the plans for final science analysis and exploitation of SKA data. This function is largely implemented by the DELIV element of the SDP.

1. The astronomer interface (F.14, SKA1-SYS_REQ-2353, SKA1-SYS_REQ-2352) provides the interface to individuals to search, summarise and obtain data from the SDP archive.
2. Regional Centre Interface (F.13, SKA1-SYS_REQ-2354 but also motivated by developments after receipt of the L1 requirements) provides the interface that allows regional centres to systematically obtain data from the SDP archive and then serve it to local astronomers.

The SDP is not responsible for making available outreach and general education type functions and data.

6.1.5 Control, Monitoring & Feedback

1. Control of SDP (F.15, SKA1-SYS_REQ-2431)
2. Obtain from TM (as metadata or in the form of a Telescope Model) and manage within SDP information pertaining to the state of the system required for the operation of the SDP processing.
3. Feedback of a specialised set of calibration solutions back to the TM so that they can be applied in real time in other elements of the SKA system (to support phasing of telescope for beam-formed observation, function identified to satisfy the architecture of the SKA).
4. Feedback of updates to the Global Sky Model back to the SKA system (F.11, satisfying SKA1-SYS_REQ-2322)
5. Feedback of information on how the observation is progressing (part of F.15). Two types of feedback will be provided: real-time visual displays that will enable the operators to visually assess the progress of data processing, the observation and identify unusual problems. And,

[2] The only other sources of information for astronomers are the alerts of transient sources that are passed from SDP to the TM and distributed to the astronomical community via the TM. These will be relatively few and containing a very small amount of information as they are primary goal is to allow follow-up observations with other telescopes.

quantitative metrics that can be used to algorithmically or heuristically make decision on whether and how to proceed with observing and if the data are likely of scientifically useful quality – see requirements SKA1-SYS_REQ-2744 and SKA1-SYS_REQ-2347.

6. To send information on status of the SDP as a system with relevant breakdown back to the TM manager and send all telescope model configuration to TM for long term storage in the global system model database (SKA1-SYS_REQ-2645).

6.2 PHYSICAL CONTEXT

The instances of the SDP will be situated in or in the neighbourhoods of Cape Town and Perth as specified in the Telescope System requirements [AD 03], in buildings specialised for housing computing and data centres. The buildings will be secure and powered from the national grid of the respective host countries. Both of the cities are large and well served by international passenger and freight flights. The cities are not located in areas with frequent natural disasters. The physical context is therefore relatively benign.

All data connectivity to the site will be provided by the SaDT SKA element. The buildings, power, HVAC, physical security and miscellaneous services will be provided by the Infrastructure SKA element.

6.3 OPERATIONAL CONTEXT

The SDP has to operate as an integral part of the SKA observatory and be scheduled and coordinated in real time with the other elements of the SKA. This is because the visibilities input to the SDP are not stored anywhere before they arrive, because the visibility storage capacity of SDP will be limited and because SDP needs to provide feedback on some calibration parameters to the rest of the SKA system in near real time.

This close co-ordination of data taking and data processing is unusual in traditional HPC facilities and data centre facilities and gives rise to important requirements on the SDP. In particular the SDP must be able to quickly configure itself to accept input data from the telescopes, it must be able to calculate in advance how long it will take to process certain observations and subsequently it must be bounded within that estimated time. Additionally in order to achieve good overall efficiency of the SKA system as a whole, it is necessary that maintenance schedules of the SDP are aligned with maintenance schedules for the rest of the SKA system.

6.4 SDP INTERFACES

The interfaces of the SDP include high-level interfaces with the Telescope Manager and the Central Signal Processor, both of which are mediated at physical and link level by the Signal and Data Transport element. These interfaces are described and defined in the relevant ICDs. The SDP additionally has an interface with the Infrastructure element to define the use of infrastructure in the SDP. These interfaces are in advanced draft states but not yet been fully signed at the time of writing of this document. The context diagram is shown in **Error! Reference source not found.**.

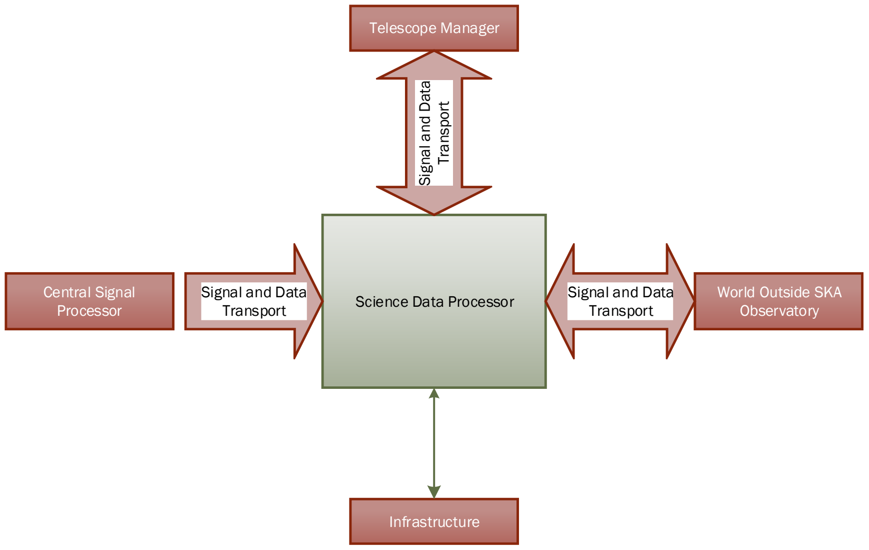
Telescope Manager
Signal and Data Transport
Central Signal Processor
Signal and Data Transport
Science Data Processor
Signal and Data Transport
World Outside SKA Observatory
Infrastructure

Figure 3: Context diagram for the SDP showing the interfaces to other elements of the SKA.

The data to be received from Central Signal Processor to SDP are described in detail in [AD 4]. They will consist of:

1. Output of the correlator: the visibility values, the flagging fractions and time centroids
2. Pulsar Search data together with associated metadata
3. Pulsar timing data including associated metadata as PSRFITS files

The interface between the SDP and Telescope Manager is described in detail in [AD 5]. In summary:

1. The TM controls all SDP functions, including querying SDP for capability to process observation data and sending the commands to begin receiving and processing observations
2. The SDP sends monitoring, quality assurance and status data to the TM
3. The SDP receives from the TM the telescope state information and sends all information relevant to the long term understanding of how the telescope is functioning back to the TM. This exchange of information logically allows the maintenance of a single, consistent, telescope model. This allows but does not require the use of a single shared, dynamic, subsystem (SKA1-SYS_REQ-2645).

The interface between SDP and SaDT is described in detail in [AD 6]. This consists of optical/electrical interface and a basic data link and framing specifications.

The interface to the local infrastructure is described in [AD 7]. This specifies the floor space, electrical power, power dissipation, cooling, physical access, and similar.

6.5 Summary of SDP Requirements

6.5.1 Scientific Performance Requirements

The performance characteristics of the SDP will have a large impact on the scientific usefulness of the SKA telescopes, especially so because in general it will only be possible to long-term store processed data. Examples of the impact are the amount of information that is retained (e.g., frequency resolution, field of view, etc), the dynamic range of the images, astrometric, photometric accuracy.

In some cases the SKA system level characteristics may be driven directly by the performance characteristics of the SDP which in turn are determined by affordability constraints. For example, in operation the SDP may be required not to archive entire spectral cubes for survey modes, but only cut-outs around identified objects in order to limit the costs of the archive. This is almost entirely determined by the affordability of archive space. The SDP architecture should be capable of supporting such operational modes.

In the majority of cases, however, the final performance of the SKA will be a complicated function of performance characteristics of many of the SKA elements and of some things outside of the observatory. The obvious example of this is the dynamic range of imaging that depends on performance of receptors, digital electronics, SDP as well as on the statistics of sources on the sky, ionospheric conditions and the prevailing RFI environment. The detailed top-down allocation of performance requirements to each element of the SKA has not yet been made, and indeed it would have been difficult to make. The reason is simply that it is not clear without some analysis what the relevant performance characteristics even would be for each of the elements.

For the current documentation we have therefore concentrated on identifying the performance characteristics that we believe are important for the SDP and/or relevant for system performance of the SKA.

One of the critical drivers of cost and risk of the SKA is the capital and operational expenditure of the computing hardware. We estimate the sizing of the compute hardware on the basis of our parametric model for SDP computational requirements [RD 5]. The inputs to this parametric model are clearly critically important to the SDP. In most cases these input parameters are not derivable from other L2 requirements. Also, in most cases, these parameters fall into the second category described above and are not easily derivable from L1 requirements. The parameter input set of the computational requirements model of the SDP will form a good basis for parameter allocation to the SDP in the future.

6.5.2 Requirements from the Physical and Operational Context

The SDP element does not currently have hard requirements on cost and power consumption. An indicative budget has been supplied to the SDP for the cost, but this is subject to some inevitable revision during the SKA re-baselining. Likewise, an only indicative power budget has been supplied. Much firmer capital, power and operational cost budgets are expected in first half of 2015 and they will be incorporated into the engineering process. In the mean time indicative budgets have been used to guide some design points as described in the subsystem design reports.

A reliability and availability budget has been supplied which would be extremely difficult for a high-performance computing facility like the SDP to meet. The current reliability budget is constructed on assumption that planned maintenance of different elements of the SKA is scheduled independently. We will be proposing a revision of the overall SKA reliability and availability

6.6 The SKA-SDP Challenge

The scale of the processing challenge in SDP-SKA has long been identified – an overview and introduction to the issues was presented at the SDP (then Software & Computing) Conceptual Design Review [RD 16].

The main challenge is that of data and computational scale and the requirements that are consequential to this. The estimated data rates and computational requirements are modelled in detail in parametric form in [RD 5]. Here it suffices to summarise that the input data rate for an SDP instance will be of order 1 – 10 TeraByte/s, the capability for floating operations will need to be in the 100 PetaFLOP/s range and around 300 PetaByte of fast buffer storage will be needed. Such capabilities are not available in any publicly accessible facility today and are expected to be challenging even around 2022 (see [RD 17] for discussion) when the full SDP platform is expected to be first required. The challenge is to achieve or get close to these requirements as cost-efficiently and as power-efficiently as possible while maintaining the overall efficiency of the SKA as an observatory. It is generally accepted in the literature that reaching these computational capabilities and data rates will require a change in the model in which we create programs and execute them. See for example [RD 18] for a review. The changes can be summarised as:

1. Increasing number of nodes (and distinct memory address spaces) by more than an order of magnitude[3]
2. Increase in number of threads per node, specifically the number of streams of processing instructions processing data in a shared memory addressed space
3. Increasing frequency of hardware failures
4. Increasing importance of network topologies

The combination of the parametric models and our Costed Hardware Concept can be used to derive some indicative numbers for all of these for the SKA SDP.

There is also a perceived challenge in the algorithms required to make reasonable quality images from the SKA, and indeed common experience has been that such challenges exist with all new instruments. The pathfinder and precursor programmes are providing valuable developments and experience in addressing relevant challenges on-route to the SK. Some examples are synthesis of aperture array tiles by LOFAR and MWA, aperture synthesis with Phased Array Feeds (PAFs) by ASKAP, high-fractional bandwidth imaging with the JVLA, offset Gregorian high-dynamic range imaging with MeerKAT and so on. The scope of the SDP work programme is focussed on design and not research: significant research on new data analysis and algorithmic approaches *is not planned.* Hence, initially SDP aims to support

[3] E.g., the Galaxy computer at the Pawsey centre has 500 nodes while the SDP Costed Hardware Concept has around 10,000 and this is achieved by greatly increasing the number of threads per node

and implement the current state-of-the-art algorithms in actual astronomical use. Experience with past telescope projects, and the requirements of high dynamic range for the SKA mean that it is however likely that new algorithm development will be required to make the best use of the SKA.

Therefore the SDP must and does recognise is that algorithmic breakthroughs **will very likely come**: from precursors/pathfinders working on real data, from independent researchers, and finally in the SKA observatory itself once the SKA telescopes are themselves commissioned and we begin learn to about how to best use them and process the data they observe. The challenge for the SDP therefore to ensure that, once available, these new algorithms can be used within the SDP without a disproportionate amount of re-engineering. The challenge of making software general enough to adopt to these new algorithms without making it too general and thus slowing down progress on the software construction is substantial and very well recognised in the software engineering industry. These considerations lead to an assumed requirement on the SDP architecture that it must facilitate on-going algorithmic development and implementation during the SKA operational phase.

A related challenge for the SDP is how the system handles hardware refreshes, which are inevitable due to limited lifetime of high-performance computers and the power savings afforded by new technologies. Designing a system that relatively easily ports a new hardware implementation, with probably a different version of the operation system and likely a different type and capabilities of the processors is a considerable challenge.

The storage space and associated requirements to store all of the image cubes that the SKA could produce is likewise extremely high and likely economically unfeasible. This means that besides the great difficulty in producing high-quality images in the first place, some of the information in them may need to be thrown away. The SKA needs as a system to decide on a policy of how much is kept. Certain SDP functions such as cut-outs and stacking allow a drastic reduction in the stored data volume while keeping most of astronomically interesting information.

7 ARCHITECTURAL PRINCIPLES

7.1 GENERAL PRINCIPLES

We have adopted the following list of architectural principles:

1. Scalability: the overall design of the SDP system should be scalable to handle a range of computational and data throughput requirements. This is in contrast to a potential architecture which aims to achieve a solution for a particular design parameter point.
2. Affordability: the SDP must be affordable, i.e., the required expenditure on hardware capital and operational costs should be close to an idealised best-case and comparable or better than the efficiency of other potential architectures.
3. Maintainability and extensibility: it must be possible for the SKA Observatory to keep the SDP software running efficiently as the algorithms for data processing evolve and the underlying hardware is refreshed.

4. Support for *current* best-practice algorithms: the SDP must support all of the current best-practice algorithms used in radio interferometry and in particular by the pathfinders and precursor instruments.

These are justified as follows.

7.1.1 Scalability

Telescopes like the SKA are obviously intrinsically scalable themselves: as additional receptors are added and as the length of available baselines increases, the scientific capability of the telescope increases, but so does the computational load. A scalable SDP would complement the naturally scalable telescopes and provide for reasonable reuse of (mostly software) investment from the early science through to full operations and finally toward SKA2.

It is hard to model the computational requirements given some performance requirements at the L2. This is in part because there isn't a complete available suite of software with all of the features required, and in part because understanding performance of current algorithms on future hardware is intrinsically uncertain and difficult. A scalable architecture alleviates some of the risks associated with this as, if the computational requirements have been misunderstood, the system can be re-scaled without re-architecting.

At the time of writing the SDP does not have fixed performance requirements – placeholder requirements have been identified and proposed at L2. The L2 performance requirements will only fully be confirmed later in the SKA system design process. Even then, it is likely that what we learn during commissioning of the SKA will cause some adjustment of these performance requirements. As a result, at time of writing, there isn't a good design point on which to try to concentrate as opposed to creating a scalable system.

Related to the above, the SKA as a whole is, at the time of the writing, undergoing a re-baselining exercise the outputs of which may significantly affect the computational requirements of the SDP. This adds to the uncertainty of exactly what a fixed design point would be.

The risk associated with designing for scalability is that excessive focus on scalability distracts from delivering a complete enough design in this phase and a working system in the implementation phase.

The counter principle would be to design for a fixed processing capacity, delivering a system which meets this before considering any scale increase. The uncertainty in the actual processing requirements, the uncertainty in future computing hardware architectures and the uncertainty in future development in algorithms makes this difficult at the present time.

7.1.2 Affordability

The argument for affordability is very simple: the capital and operational costs of the SDP are a large fraction of the respective costs of the whole SKA system. Small changes in these costs impact significantly the costs and therefore the scientific capabilities of the entire instrument.

The counter principle would be design for simplicity, correctness and full functionality first and incrementally improve the affordability of design during construction. The attraction of this would be a reduced execution risk in the project plan during construction. The disadvantage that this would mean we do not design-in the lessons learned from precursors and pathfinders on what makes efficient systems and that this would require far more total development work during construction then a designing for efficiency right from the outset in the design phase.

7.1.3 Maintainability and extensibility

The operational cost of electrical power for the SDP means that the hardware refresh cycle is going to be relatively rapid so to make best use of increasing power efficiency of computing hardware. It is important that the investment in the SDP software over these hardware refreshes is largely maintained and that means a reasonably maintainable system is required.

Furthermore, experience with SKA pathfinders and other radio telescopes has shown that once the commissioning and scientific observations with new telescopes start it was often the case that new algorithms and techniques are identified that can correct some unexpected effect in the telescope system, in existing algorithms or in the atmosphere. Therefore, it is expected to be useful that it is possible to add to the functionality of the SDP with reasonably limited effort.

Additionally, the design and construction of phases of SDP are sufficiently long that the maintainability will become important even before the SDP system reaches full operations. For example, the time period between the time of writing and the beginning of construction (2015-2018) is *shorter* than the anticipated time between the beginning of construction and full deployment of the SDP (2018-2022!).

7.1.4 Support for *current* best-practice algorithms

It is very likely that some novel algorithms will be required to get the best possible science of the SKA telescopes. It is not, however, the purpose of the SDP design project to be researching these new algorithms – the current SDP design phase has to focus on delivering a workable and complete design and that precludes reliance on not-yet-discovered algorithms. At the same time, the SDP processing will be very challenging and the investment in SKA very large. Therefore the SDP will support an appropriate selection of current best-practice algorithms for interferometric data reduction tailored for the SKA functions and requirements.

A critical corollary is that iterative algorithms must therefore be supported and this is a major design driver for the top-level SDP architecture.

7.2 Data Parallelism and the SDP

It is an inevitable consequence of the SDP's large processing requirement [RD 5], the large input data volume and the fact that most of the processing is concentrated in a few key algorithmic steps (e.g., gridding, FFTs) that the SDP implementation will have to be largely *data parallel* [RD 4][4]. By this we mean that the basically same processing step will need to be executed concurrently on many different input data[5] (from same parent observation), producing intermediate outputs which are later combined as necessary. A simple example of data-parallelism is gridding of visibilities onto multiple *uv* grids simultaneously which are later combined. Here the same operation (gridding) is being executed on different data (visibilities, e.g., belonging to different frequency channels) simultaneously.

The very large data volume associated with each SKA observation precludes the possibility of the simple and embarrassingly-parallel strategy of reducing many observations concurrently. If this were possible, the parallelism of the processing of each observation would be proportionally smaller, reducing the demands on the network interconnect of the computer cluster and reducing the software complexity necessary to achieve parallelisation with reasonable efficiency.

Because of the inevitability of data-parallelism it is essential that the SDP architecture is well suited to appropriate data-parallel algorithms and that the design of the processing pipelines will identify and/or anticipate data-parallel stages and carefully analyse non-data-parallel stages for scalability.

7.2.1 Illustration of data parallelism for visibility processing

In this discussion we present an illustration of processing of visibility data output from the correlator and its processing into images as this drives the overall compute cost and system scaling. The discussion here is illustrative, the full analysis of data distribution strategies is in the Pipeline Design document.

The problem we must solve is simplified compared to a generic problem since we have essentially two shapes of data:

1. Visibility data which can be thought of as a partially ordered stream of data values indexed by position (*u,v,w*), polarization (or correlator product), frequency (*f*), and time (*t*), or alternatively baseline (antenna pair), polarization, time and frequency. A hybrid indexing between these descriptions is possible and is likely to be exploited for, for example, snapshot imaging where the natural indexing is (*u,v,w,f,t*).
2. Rasterized data in which each pixel/voxel in gridded *u,v,w* space or in image space is further indexed by polarization, frequency or potentially time. Further indexing of rasters may be by, for example, position (in faceting imaging).

[4] An example of parallel execution which is **not** data parallel is pipelining, where there are different stages of the computation that are executing simultaneously on data which they received from the previous stage. The top-level sequence of operations to be applied to observed data to produce science ready data product is and is normally called a `pipeline' in radio-astronomy literature but its depth is insufficient to give the required level of parallelism.

[5] "*Perhaps, in retrospect, this is a trivial observation in the sense that, if the number of lines of code is fixed and the amount of data is allowed to grow arbitrarily, then the ratio of code to data will necessarily approach zero*", from [RD 4]. In the case of the SDP the input data volume is about two orders of magnitude greater than LOFAR and ASKAP.

The key concept behind the architecture is that the data layer should be *able* to exploit any route to data parallelism. Of course the actual data parallelism used in practice will be balanced against the additional costs of employing it (e.g., due to data duplication, non-sequential access of data, etc).

In practice this means:

A. Distribute visibility data to a data locality based on any defined set of criteria on the index of the visibility sample. Data duplication is permitted so that the same data are sent to multiple data localities.
B. Distribute any region of rasterized data to a given data locality: again data duplication is permitted either by totally duplicating a region of the raster or partially to implement, for example, guard-regions around a cut-out from a raster.

Data duplication allows a greater range of parallelisation strategies without the need for unified storage. It however inevitably has some costs in power and storage requirements. The scheduling and data duplication strategy that will be chosen will balance these additional costs versus the improvement in execution efficiency and potentially reduced network traffic that finer grained parallelism allows.

Splitting the data on some indices of the visibility data will be far easier than on other because of the regularity of the index and the natural grouping of the data when output from correlators. For example splitting the data according to frequency is likely to be relatively easy while splitting the data according the *uvw* (which amongst other things are functions of the frequency) significantly harder. The choice of splitting strategy will need to balance the benefits against actual additional costs.

Of course the ways in which the data are distributed to exploit such inherent data parallelism will not be arbitrary, but will depend on the specific processing to be applied to the data. Also the specific data distribution approach taken needs to take into account the characteristics of storage and network transport systems: these tend to be far more efficient when handling large chunks of continuous data then when handling many small read/writes or messages.

The architecture implements this by defining a logical data-flow graph describing the requirements for how the data are to be distributed and where it is needed for a given pipeline.

Examples of specific ways in which the data may be distributed depending on pipeline are:

- Division of the visibility space:
  - Main index is frequency+polarisation: then
    - Snapshots (rotated planes in *uvw*) -
    - *w*-stacking (slices in *uvw*)
    - Faceting (duplicate visibility data)
- Division of image space
  - Main index is frequency+polarisation: then

        - Image-plane facets
        - Division of spectral cubes into 3d cut-out volumes for source searching including guard regions
- Time: accumulate visibilities into separate grids and combine later
- Beams

Simplified examples are illustrated in the following figures.

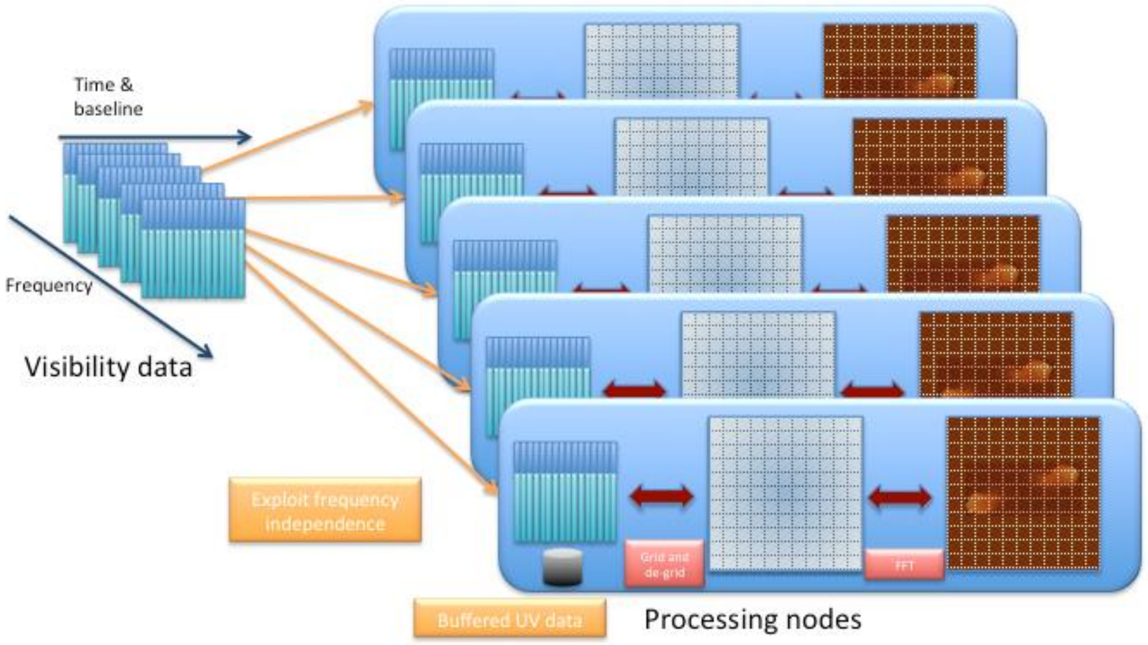

Figure 4: Illustrating data parallelism based on frequency selection only (c.f. LOFAR)

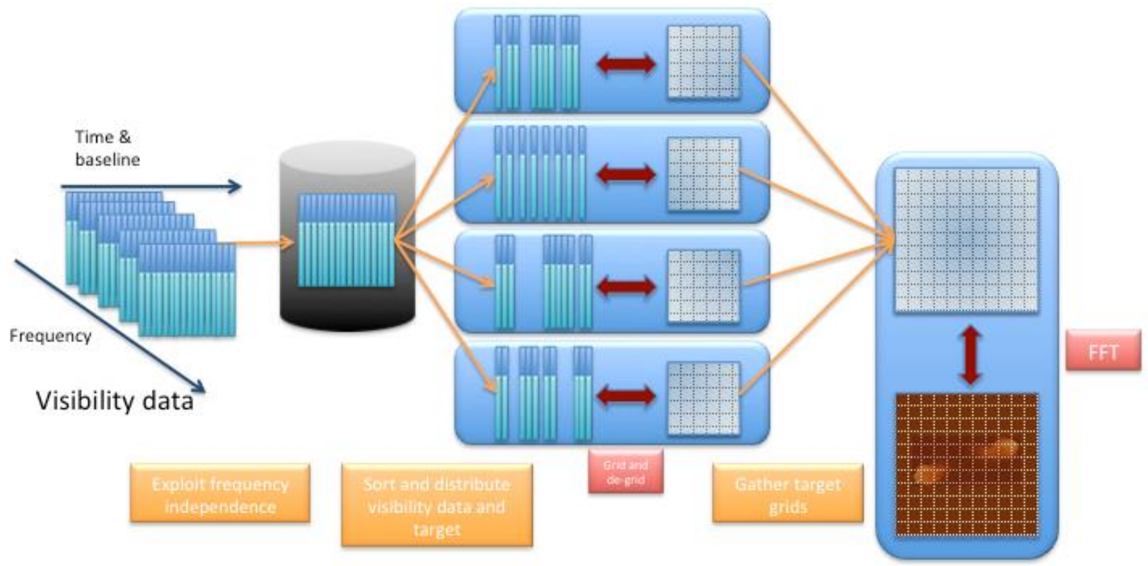

Figure 5: Managing memory bandwidth by distributing regions of target uv-space over nodes followed by a gather and FFT to image space.

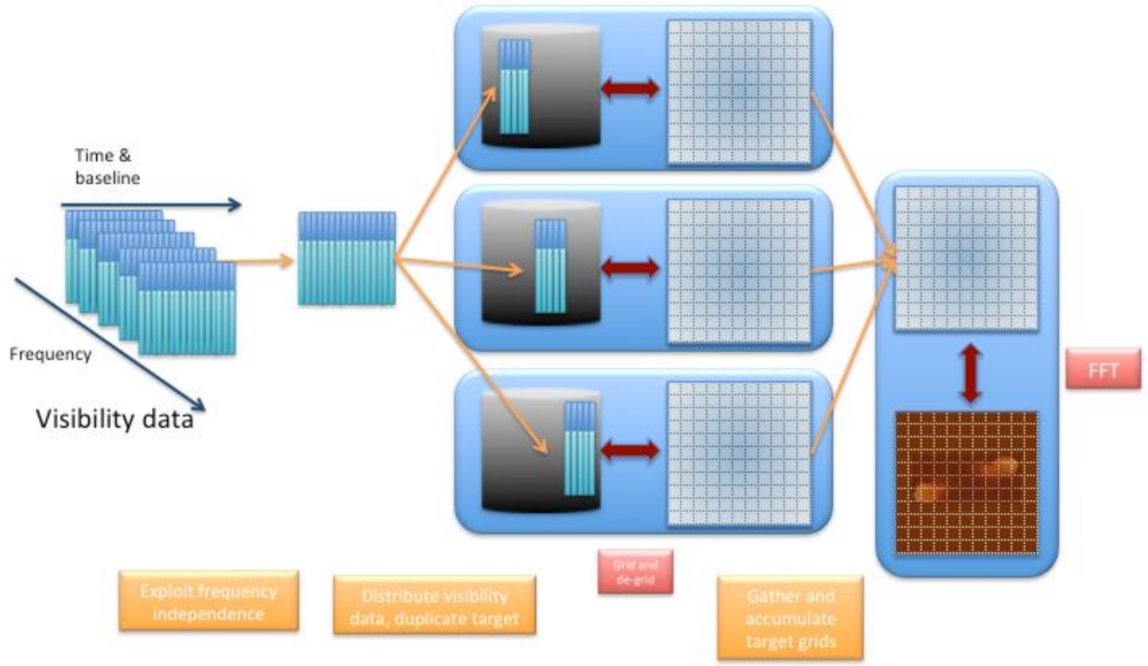
Time & baseline
Frequency
Visibility data
FFT
Exploit frequency independence
Distribute visibility data, duplicate target
Grid and de-grid
Gather and accumulate target grids

Figure 6: Managing IO bandwidth from local storage by distributing source data over multiple nodes. Target gridded visibility plane is duplicated then an accumulation step is performed over the data island over which the source data were distributed

8 OVERVIEW OF ARCHITECTURE

8.1 SDP ARCHITECTURAL DECISIONS

The SDP-wide design decisions are presented and justified here, ahead of presenting the decomposition of the SDP system that is in part motivated by these decisions.

8.1.1 A partially near-real-time and partially buffered processing model

8.1.1.1 Motivation

Some of the SDP processing requires outputs which are near-real time: currently identified functions with this requirement are F.4 Real Time Calibration, F.7 Imaging Transient Search and F.8 Non-imaging transient post-processing. The processing of the remaining functions is not time-critical and typically requires iterative access to the data.

8.1.1.2 Description

The SDP will receive data from the CSP promptly after they are observed. In the case of visibilities from the correlator this will be as a continuous stream without possibility of pause or retransmit, while in the case of data from the non-imaging processor there will be some capability for flow control. These data will then be merged with a sub-selection of Telescope State Information data (received separately from Telescope Manager) that is required for the next processing stages. At this stage data will also be flagged for RFI, strong sources will be subtracted and initial averaging applied to reduce the data rate when possible.

The same data will then be used for both near-real-time and batch processing. The batch processing will be enabled by a buffer system that records data from the ingest and stores until the observation is complete, and iterative processing of the whole observation can begin.

In the simplest case the buffer will operate in the double-buffered mode: while one observation is being recorded on the buffer the batch procession of the previous observation will be carried out. However, it will also be possible to record more than observation in the buffer for later processing in order to enable some load-balancing to be done between a series of observations. This is illustrated in Figure 7.

8.1.1.3 Justification

The continuum imaging and spectral line imaging functions require iterative access through the visibilities, and therefore a buffered system is required to support them. Other functions require near-real-time outputs as described in the motivation. Therefore it is unavoidable that the SDP adopts such a processing model.

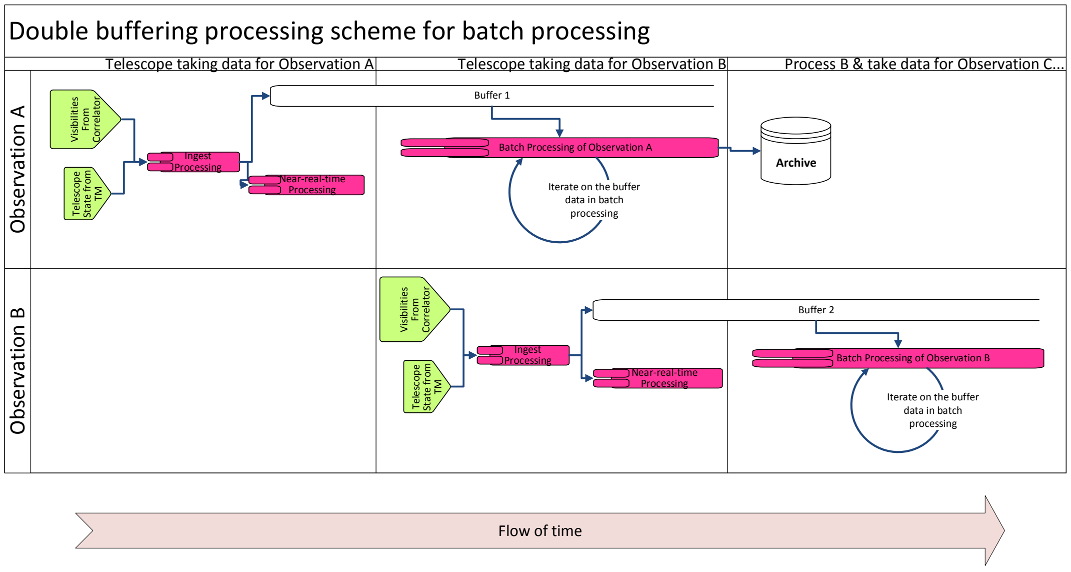
Double buffering processing scheme for batch processing
Telescope taking data for Observation A
Telescope taking data for Observation B
Process B & take data for Observation C...
Observation A
Observation B
Visibilities From Correlator
Telescope State from TM
Ingest Processing
Near-real-time Processing
Buffer 1
Batch Processing of Observation A
Iterate on the buffer data in batch processing
Archive
Buffer 2
Batch Processing of Observation B
Flow of time

Figure 7: Illustration of the double buffering scheme for batch processing

8.1.1.4 Implications

The key implications are:

- The requirement for a buffer system which will require a high capacity and high input/output performance
- The requirement that some pipelines can operate in near-real time without relying on the whole observation being completed
- The data from the ingest stage has to be passed to both the buffer system and the near-real-time processing while minimising extra data copying and movement. This for example means that the splitting of data for parallelisation has to be optimised simultaneously for the batch and near-real-time processing functions.

8.1.2 Dataflow programming model for the SDP

8.1.2.1 Motivation

The motivation is primarily that of *separation of concerns*, i.e., we want to separate as much as possible the specification of the details of the data manipulation and reduction that needs to be done, from the way this is scheduled on the compute system, how it is sequenced and how the associated data are transferred and stored. We are further motivated by the success of the popular *Map-Reduce* methodology [RD 3] as for example implemented by *Hadoop* [RD 8], the data processing environments like SWIFT/T [RD 13, RD 14] and the high-performance linear algebra library DPLASMA (built on DaGue/Parsec) [RD 12] which all use elements of the dataflow model.

8.1.2.2 Description

Dataflow programming contrasts with the traditional (von Neumann) programming model in both the concept of control flow and memory. [RD 11] is a recent review of dataflow programming. For the SDP we adopt the dataflow model at the ***macro*** level, i.e., for relatively large components of the system. The model of computation inside the coarse-level *actors* or *tasks* is not specified at the architectural level. This specific approach is called *coarse-grained* dataflow, see for example the review [RD 19].

Additionally, although the dataflow model will apply to the major part of the SDP, i.e., the part that is responsible for the main data processing functions, the dataflow model will not apply to all of SDP. In particular the local monitor and control will be outside the dataflow model (in fact one of its functions is to control the dataflow system), and some data handling will be outside the dataflow model: this includes the handling of telescope state and calibration data (which will be done in via the "local telescope model") and of the monitoring data streams.

In the dataflow model there is no explicit specification of the control flow; instead computation is done at some time after all of the inputs required for computation are available (whether from an external input or by computing an intermediate result). For example, a frequency averaged image may be computed by combining channels 1 to N. In the dataflow model channels 1 to N would be specified as inputs to the combination calculation and the run-time system would then schedule the computation of each of the channel images (in some arbitrary order) before the combination step. In contrast, in a traditional model, one would explicitly specify to first compute the channel 1, then channel 2, and so on up to N by looping from j=1 to j=N and calculating image for channel j and then specify the combination step.

It is a necessity of dataflow programming that each computation explicitly states its input and output. Each computation can only access these input and output values and some "local" working memory. The local working memory is not accessible to any other computation. This property is sometimes termed *no side-effect* as the only changes to the environment seen outside each computation is through its output values. For example a minor CLEAN step would not be able to directly change a global sky-model in the dataflow model. Instead the CLEAN would produce a list of additional components found which would be merged into the master list of components at the appropriate point in the cycle.

8.1.2.3 Justification

The primary reason for adopting the dataflow programming model is to allow efficient scalability to large number of nodes within a feasible amount of programming effort. The advantages of the dataflow model in comparison to others:

i) It is much easier to load-balance a dataflow model because the schedule of computation is specified independently of the application. This load balancing can be static (where it is determined at the beginning of each run), or dynamic when the schedule is constructed and adjusted during run time of processing, or a mixture of the two.
ii) It is easier to ensure data locality in a dataflow model program because each step explicitly specifies its inputs and outputs. Data locality impacts the efficiency especially in systems with inhomogeneous communication costs.
iii) The actual flow of control of sub-graphs of the dataflow can be determined without any reference to other partitions of the graph, reducing the throttling of scalability by master nodes in master/worker type control flow system.

Dataflow system implementations today are already capable of handling and executing programs of complexity comparable to, and greater than, SDP. For example if an SDP input dataset of 100 PetaByte is divided into 10 GigaByte chunks and these each comprise about 50 stages of processing the SDP dataflow program will consist of about 500 million tasks in total, probably to be executed over several hours. The SWIFT/T execution engine is capable of initiating and controlling over 1 billion tasks *per second* in dataflow programs with data dependencies. DaGue/Parsec is today used in production and has fully independent control flow where after initial partitioning of the graph control flow proceeds fully independently of any central master.

The dataflow model is restricted to the major data processing part of the SDP where it makes the best fit. Telescope state, calibration parameters and sky models are not covered by this model because of the complexity of including them and the fact that alternative models for their maintenance can be constructed based on their relatively low data volume, low update rates, and clear and simple update semantics (the calibration parameters and sky models are updated precisely at the end of a calibration/clean cycle).

A data distribution and dependency diagram for the continuum pipeline is shown in Figure 8. Although this is not a full dataflow program or graph, it shows the flexible partitioning of data that is possible and the decomposing of the processing steps into simple components with reasonably compact interfaces and which could be made referentially transparent.

The dataflow approach also provides a clear partitioning of the software stack:

The dataflow layer exploits the inherent data parallelism and is responsible for interfacing to system resources and delivering the required data intensive scaling performance needed. Domain specific characterisation is built into the data model and especially how the intrinsic data parallelism is exploited but also efficient deployment of data needed by processing components.

The processing components within the data flow encapsulate the majority of the domain-specific implementation required to deliver the overall algorithmic pipeline. However the structure of the components themselves does not implement the overall system scaling. It is therefore a significantly less difficult data-centric-HPC programming task to implement a component. Other architectural decisions on components are discussed in 8.1.5.

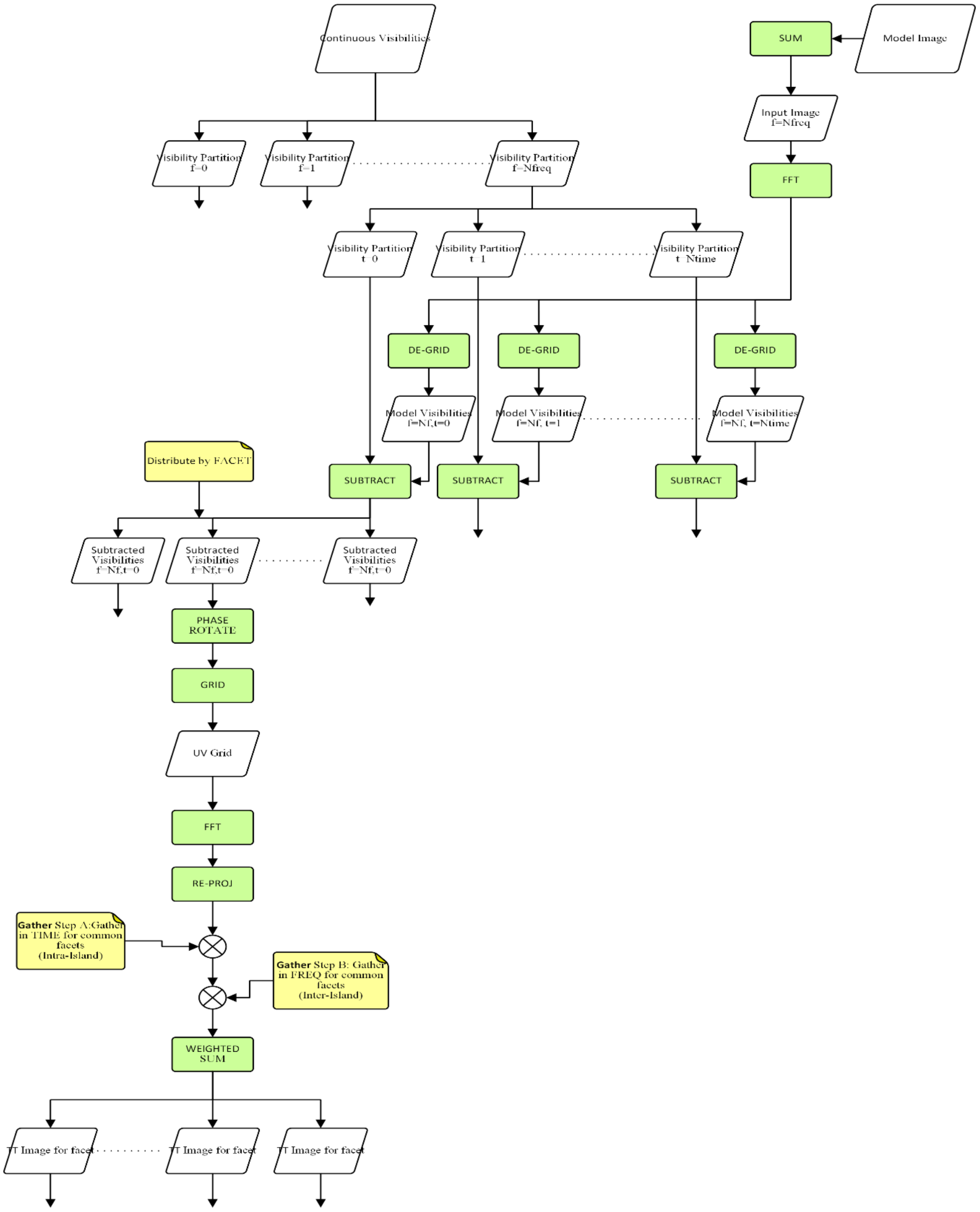
Continuous Visibilities
SUM
Model Image
Input Image f=Nfreq
Visibility Partition f=0
Visibility Partition f=1
Visibility Partition f=Nfreq
FFT
Visibility Partition t=0
Visibility Partition t=1
Visibility Partition t=Ntime
DE-GRID
DE-GRID
DE-GRID
Model Visibilities f=Nf,t=0
Model Visibilities f=Nf, t=1
Model Visibilities f=Nf, t=Ntime
Distribute by FACET
SUBTRACT
SUBTRACT
SUBTRACT
Subtracted Visibilities f=Nf,t=0
Subtracted Visibilities f=Nf,t=0
Subtracted Visibilities f=Nf,t=0
PHASE ROTATE
GRID
UV Grid
FFT
RE-PROJ
Gather Step A:Gather in TIME for common facets (Intra-Island)
Gather Step B: Gather in FREQ for common facets (Inter-Island)
WEIGHTED SUM
TT Image for facet
TT Image for facet
TT Image for facet

Figure 8: Data distribution diagram for the Continuum Imaging function. Green rectangles show processing steps (i.e., dataflow actors), lines show data dependencies, slanted rectangles describe the intermediate data. See [RD 27] for the complete description.

8.1.2.4 Implications

The most obvious implication is that the SDP program at the top level will have to be specified in a dataflow language, which eliminates all of the commonly used computer languages in radio astronomy (Fortran, C++, C, Python).

The processing tasks/processing components/macro actors internally can use any computer language but each component has to be free of side effects. This can also be expressed by saying that the processing components must be referentially transparent, i.e., their outputs can only depend on the explicit inputs they've received and cannot depend on their internal state. This is in direct contrast with object-orientated programming typical in CASACore, CASA and ASKAPSoft which emphasises a mutable state encapsulated by the object.

Each of the processing components will have to exhaustively specify its inputs and outputs.

The components will have to be stateless so that they can be used anywhere within the processing graph.

The SDP will operate multiple functions simultaneously (e.g., real-time calibration and continuum imaging). These functions may have inter-dependencies, will share compute islands and will share input data (i.e., there won't be a separate copy of visibilities for each). For this reason, it is necessary to include all of the functions (pipelines) into a **single dataflow graph** for scheduling.

8.1.3 Non-uniform communication between nodes

8.1.3.1 Motivation

The SDP compute hardware system [RD 25] is likely to have a design with compute island which have low communications cost internally and a higher cost inter-island. Here we use the term cost to refer to the computational overhead of communication: i.e. maximum throughput capacity, latency and potentially power consumption. Low communication cost therefore implies a high-bandwidth physical interconnect, with low latency and with a topology approaching that of all-to-all connection and vice versa for high cost communications.

Making explicit the fact that the cost of communication between nodes of the SDP will not be uniform will allow the design to directly tackle the non-uniformity of communication and address these from the start at the application level. This type of architecture is similar to that employed within existing data intensive processing system today (such as Hadoop).

The motivations for non-uniformity are two-fold:

- Scaling of a low communication-cost architecture to the scale of the system required represents a very high risk to achieving a scalable architecture. This is evidenced by the existence today of non-uniform communication costs in existing HPC and data centre architectures.
- The financial cost of the network fabric is directly related to the scale of the system (i.e. number of nodes) over which low communication-cost is required.

8.1.3.2 Description
The SDP communications between nodes will have non-uniform costs, i.e., for each node there will be a (probably small) set of nodes with relatively low communication cost and a larger set with higher costs. By costs we mean maximum throughput capacity, latency and potentially power consumption. The design of the SDP will take this into account at the SDP application level, ensuring data locality as far as reasonably possible. The optimisation of parallelisation and data distribution will be done statically, once per observation, by the SDP software system.

8.1.3.3 Justification
[RD 21] shows that a full fat-tree[6] is economically unattractive for the SDP. Existing large computers with commodity nodes already tend to have non-uniform topologies such as tori and dragonfly (see, e.g., [RD 20] for measurements on the Cray Aries interconnect). Another example are the IBM BlueGene series of computers with a torus network topology.

Very non-uniform communication is already widespread in commercial data centres and was one of the reasons for development of map-reduce technologies (see [RD 3], Section 3, item (2)!).

The MPI Standard Version 3 [RD 22] supports specification of application topology between ranks precisely in order to be able to optimise for the actual network topology and non-uniform cost.

Taking into account the network topology is therefore already either accepted or standard practice for large computing installations either data-centric or HPC.

Exposing this topology at the SDP application allows SDP to do application-specific optimisation in order to make best use of it. Because the dataflow specification makes explicit all of the communication requirements between the processing tasks, and because these are known at the time of the setup of the observation, this optimisation should be reasonably straightforward.

8.1.3.4 Implications
The primary implication is that the use of global shared data structures (such as general purpose file shared file systems) has to be minimised since they hide the locality of the data from the application. A greater degree of message passing within application has to be envisaged, or the use of frameworks that are aware of data locality and non-uniform costs.

8.1.4 Non-precious data as a route to high overall availability

8.1.4.1 Motivation
In a large system formed of many nodes there will be inevitably be frequent total failure (computation never completes) or over running tasks (computation takes much longer than on average) on some node in the system. Dealing with such errors at component or node level can be very expensive. The structure of the SDP processing problem means that a much simpler model for dealing with failure can be

[6] "Full" to distinguish it from "Pruned" Fat Trees. Full Fat Tree is a network topology with uniform bandwidth between any two sets of nodes.

adopted for a large part of the computation using the concept of non-precious data which does not require high availability and precious data which does.

8.1.4.2 Description

The Science Data Processor uses the concepts of “Precious” and “Non-Precious” data. Non-precious data are those data which can be lost, corrupted or never computed without jeopardising the scientific integrity of the whole observation. Precious data are all other data, i.e., data which are required to ensure scientific integrity of the final data products, or which are required for completing the processing.

The concept of non-precious data can be applied to input data, intermediate data products and output data products. In the case of input data, precious data are for example key parts of the telescope state information. If some of these key data are missing and cannot be re-transmitted from the TM then SDP has to signal an error to the TM rather than continuing with the observation. An example of non-precious data are the packets containing visibilities: these can be dropped generally only with a slight loss of signal to noise in the output results. An example of non-precious data in the intermediate data products are images from a single frequency channel; depending on the observation these may lead a reduced signal to noise in output continuum images or a flagged frequency channel in the output spectral cube. An example of non-precious data in the output data are the data associated with a frequency channel in the output cube. If this is missing it can be marked as flagged.

8.1.4.3 Justification

Although typical computer components have relatively long mean time between failures the SDP will have a very large number of such components. Previous failure analyses of such systems [RD 23] have shown relatively short times before one of the components fails and it is therefore now commonly accepted that graceful handling of failures is essential for the next scale up in computing systems. Increasing the mean time between failure of individual components is uneconomical as the purchase price tends to increase very rapidly with increasing reliability specifications.

Handling **all** failures by conventional “high-availability” (HA) techniques which ensure all data are recoverable in cases of individual failures of nodes and all computations can be repeated if necessary is undesirable for SKA SDP for several reasons:

1. Costs of traditional HA are significant, sometimes leading to a doubling of the hardware cost
2. In a networked system such as SDP that will have a partially oversubscribed overall network, implementing traditional HA can be difficult because retransmitting data between islands will place additional load on the network which may slow down other ongoing tasks, leading to cascading degradation of system performance
3. Since conventional HA restarts failed processes (possibly on a different node or island) it impacts the determinism of time-to-compute. If used pervasively would require additional margin in estimated time-to-compute, potentially leading to reduced overall efficiency of the SDP system

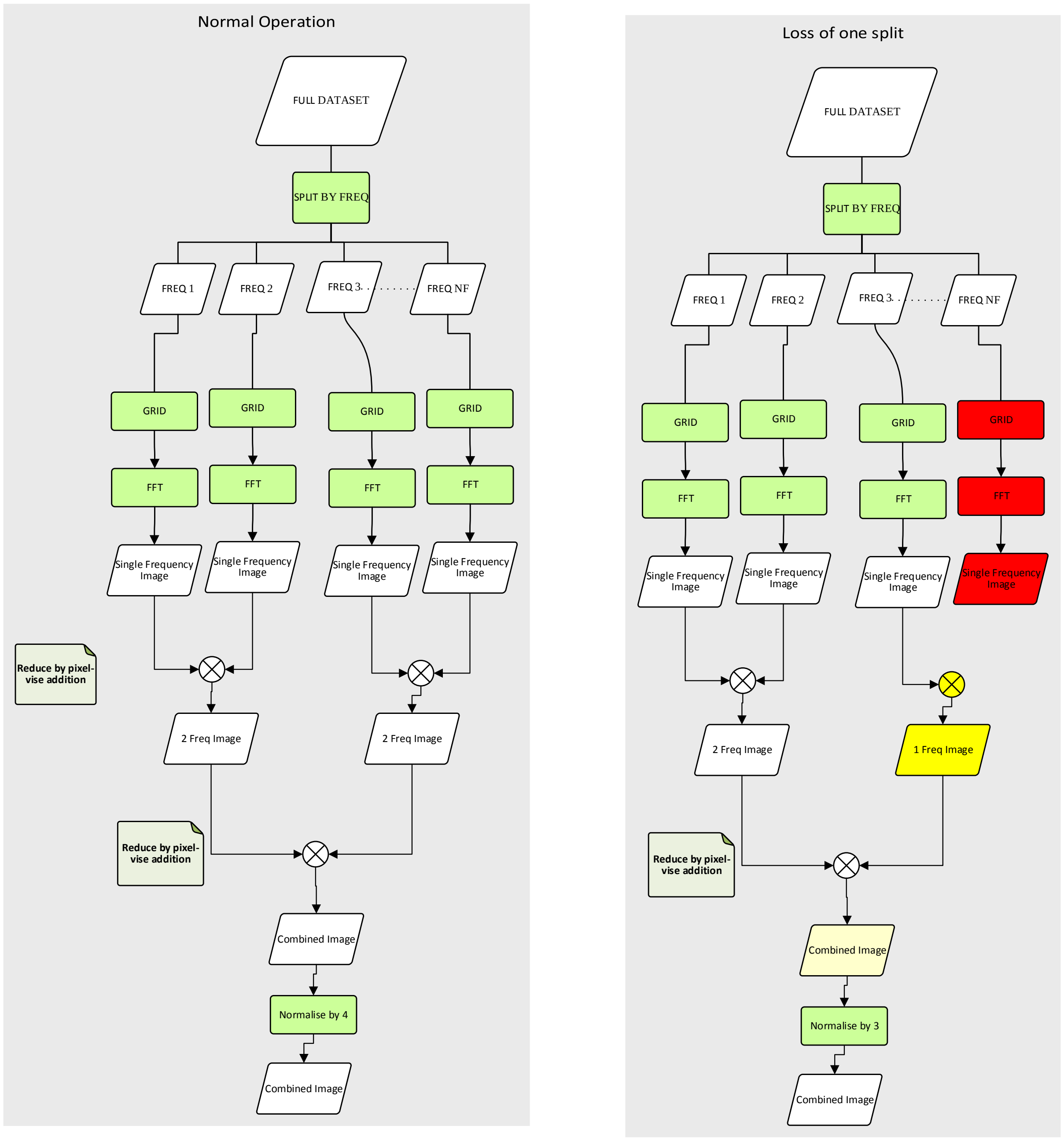

Figure 9: Illustration of the Non-Precious data concept. Arrows show the flow of data, rectangles typical processing operations and the crossed circle a reduction-type operation. One the left is a normal dataflow in which images from four frequency splits are combined pixel-wise to form the average. On the right is an illustration of what happens if there is a problem in the calculation in one of the frequency channels. All of the operations before the reduction (red) fail completely and there is no output. However, in the first reduction (yellow) there is still an output but with reduced signal/noise statistics and similarly for the final reduction.

Since SDP processing is largely data-defined, data-parallel and data-driven, it is advantageous to mark data as precious or non-precious rather than, for example, individual processing steps.

Additionally, many distributed computing applications exhibit the problem of “stragglers”, i.e., a very small number of tasks that take very much longer than expected to complete. Often this is due to problems in the I/O, network or operating system components, but sometimes due to application programming errors. In the case of SDP these need to be treated as failures. By using the concept of non-precious data such stragglers will be dealt with easily if non-precious while any tasks producing precious data can, in the extreme, be duplicated so that impact of stragglers is minimised.

Implementing the concept of non-precious data in SDP would allow graceful handling of failures by degradation of output result quality. This fits well with the SKA definition of availability and would also allow of some cheaper components within the SDP.

Multiple levels of preciousness were considered initially. This option was rejected because the availability of the telescope is defined largely in terms of the signal to noise ratio of the output data products. Therefore dropped non-precious data may only lead to small reductions of signal-to-noise in final product and so the advantage of multiple levels of preciousness would be very small.

An example of similar concept of non-precious data applied to very large scale Monte Carlo applications is described in [RD 24].

8.1.4.4 Implications

It is expected that the majority of SDP computational components would **not** handle the concept of data non-preciousness. They will either be run if all of their input (both precious and non-precious) data are available, or if some the non-precious data are missing they will simply not be run and all of their output data products likewise marked as missing.

The processing tasks implementing *reduction* and *filter* operations will however have to handle missing data. The archetypical examples are:

- The computational component that averages multiple grids into a single grids. Missing non-precious data here corresponds to a missing input grid; this is handled simply by not including it in the averaging process
- The data manager operation that combines several frequency channel images into an output image cube. A missing frequency channel image is flagged in the output cube

The implications for the SDP system are:

1. Introduction of non-precious data concept means that there will be additional uncertainty in the signal to noise characteristics of SDP science data outputs and also that some output data may be flagged. Given the typical effects of RFI and calibration errors and the uncertainty in predicting these, the additional uncertainties due to the non-precious data model are unlikely to be of significant effect on further scientific analysis of the data.
2. Pipeline definitions will have to declare which input, intermediate and output data are precious
3. Dataflow manager will have to understand the concept of non-precious and precious data. Precious data will need to be processed with suitable alternative HA model.
4. Data Managers will need to understand the concept of non-precious and precious data and to:
    a. Handle missing non-precious data

    b. Terminate over-running tasks and declare their outputs missing
    c. Implement HA strategy for precious data
    d. Flag outputs according to missing data
5. A (expected to be small) number of computational components will need to handle missing non-precious input data. These components need to produce outputs even when some of the input data are missing.

8.1.5 Design for computational components

8.1.5.1 Motivation

Components implement the processing stages of the pipeline within the context of the dataflow approach. A requirement on a component is that it is re-usable anywhere within the dataflow system and the architecture for components must ensure the scalability of the overall system. However it is also required that the component be computationally efficient and operate within an environment in which a computational node may have a very large number of cores and indeed a given component instance may span multiple nodes within a data island.

Many of the key computational steps in the SDP require significant amounts of very fast working memory (e.g., gridding, FFTs, see [RD 5]). Although there are techniques such as faceting which reduce the working memory requirements current analysis of these are best applied only to a moderate degree [RD 5]). Working memory requirements are for example one of the current challenges of ASKAPSoft, which is a single threaded code working on moderately multi-threaded nodes. Designing the computational components to be capable of efficient execution with many parallel threads is one way of reducing the total working memory requirements while achieving high throughput rate for computations.

The trend to hybrid programming with multi-threaded and message passing memory models is already evident in high-performance computing (e.g., see [RD 18]), in most enterprise computing and at a moderate level in consumer computing. As limits of frequency scaling of processor has been reached this trend of increased threading capacity is expected to increase in the future.

8.1.5.2 Description

By multi-threaded we mean that we mean that multiple threads of execution can efficiently and concurrently access a single shared working memory. The threads may have fully (Multiple-Instruction Multiple-Thread) or minimally (Single-Instruction Multiple-Thread) divergent control flow but for many algorithms will not be necessarily accessing adjacent data in the way vector processing units thus differentiating this concept from single-instruction multiple-data (SIMD). The multi-threaded capability maybe implemented by using a general language such as OpenMP that maps to many architectures or by using more specialised languages.

The alternative decision to this one would be to design the parallelisation options so that the SKA SDP problem is broken down into sections which require small amounts of working memory and therefore enable very many cores to be applied to the processing without multi-threading. While this may indeed be possible the initial analysis does not suggest this is the case and for reasons of reducing risk the multi-threaded capability should be designed-in from the beginning of the detailed design phase of computational components.

Other considerations for the component design so that it may operate within the dataflow environment are:

- It is re-usable anywhere in the dataflow system
- Communication between instances of the component must be highly efficient and therefore can only be implemented via the dataflow system itself

8.1.5.3 Justification

Although there are clear ways of breaking the SDP imaging problem into smaller sub-problems with relatively simple interaction, such decomposition is known to have inherent costs. For example, the parametric models of computational requirement show that the faceting technique begins to increase the total required FLOP rate around 10x10 facets. Additionally, as the number of facets increases the input/output rates to storage and the message passing rates between processors handling different facets increases. Similar analysis applies to most other stages of imaging and parallelisation techniques. For this reason the SDP can not be sure that decomposition of the problem into pieces small enough that single-threaded processing components can efficiently work on them can not be assured.

Additionally, the current costed hardware concept presented in [RD 21] is based on accelerators that require a massively multi-threaded programming model.

8.1.5.4 Implications

The obvious implication is that the processing components must be designed and implemented with multi-threaded in mind. This means that during the design phase multi-threaded languages and libraries must be thoroughly examined. Local multi-threaded data structures need to be designed. In implementation phase programmers with experience of multi-threaded software engineering will need to be used.

Taking into account all of the above the implications for components are:

- They must be implemented with muti-threading in mind
- They must be stateless to enable re-use
- Communication between component instances is via the dataflow layer

8.1.6 Sub-arraying and other concurrent operation through independent SDP capabilities

8.1.6.1 Motivation

The SDP is required to be able to handle data from up to 16 independent sub-arrays (SKA1-SYS_REQ-2127, SKA1-SYS_REQ-2264) for each of the telescopes. Each sub-array may have an independent engineer or science goal and different requirements on the type processing. The SDP needs to be scalable in the sense that up to 16 separate sub-arrays can be processed without overloading the SDP elements shared between them.

The same approach will be used to support concurrent analysis if SDP resources permit this. For example the concurrent analysis of imaging and time-series data.

This architectural decision also permits other concurrent observing modes to be supported which are not currently the subject of requirements but which may emerge from further system-level analysis.

8.1.6.2 Description

Each SDP instance will be capable of providing multiple capabilities simultaneously that will be able to operate independently while sharing a single computing platform. Sub-arraying will be supported by assigning a capability to each sub-array.

The independent capabilities will be implemented by making use of the computer cluster scheduling system to allocate hardware resources for each capability, by having a flexible ingest and data transport layer that is able to deal with outputs of the correlator when sub-arraying and send the data to correct location for the capability handling the sub-array; and by having most of the control and monitoring components as well as some data transport system fully independent for each capability. The functional elements which are shared and which are independent for different capabilities are shown in Figure 10.

8.1.6.3 Justification

The primary justification for the above approach is that it reduces the complexity of most of the software components because they do not need to be aware that multiple sub-arrays are being processed. At the same this approach is practical because:

1. Computing cluster schedulers allow good partitioning of clusters and segregation of multiple processes running on them
2. The dataflow programming model allows efficient adaption of the pipeline program to the hardware being used
3. Use of an ingest layer and flexible and configurable data transport network allows the re-ordering of data so that most of the processing hardware for the capabilities can be separate.

The alternative would be to require the top-level pipeline programs and most of the components to each support multiple capabilities and to execute them in parallel. This would allow potentially better efficiency but at the cost of increased software complexity and decreased run-time flexibility, e.g., in this scenario it could be very difficult to terminate individual capability without affecting the other capabilities running at the same time.

8.1.6.4 Implications

The primary implications are the requirements on functional elements which remain shared between the capabilities:

1. The network interconnect system
2. Master Controller part of the LMC
3. Some parts of the Data Layer (if some efficient data exchange between capabilities is required)
4. The cluster scheduler

Each of these will have to be capable to deal with multiple independent capabilities in an efficient and scalable way.

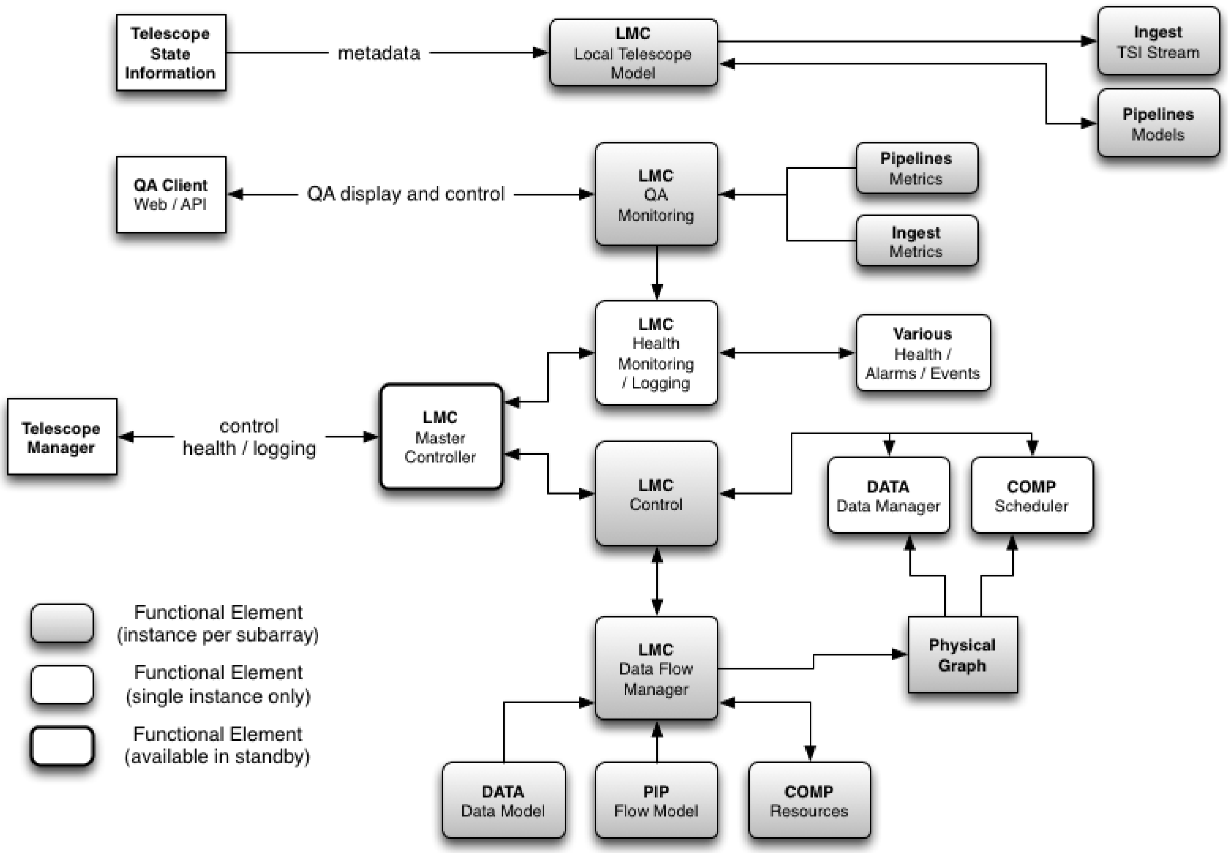

Figure 10: Illustration of the LMC functional elements showing which elements are shared between sub-arrays and other capabilities and which elements are independent. See the LMC architecture document [RD 29] for further detail.

8.1.7 Data Organisation and Data Structure

8.1.7.1 Motivation

Data organisation and data structures within the SDP must support the dataflow approach and take into account the other structures imposed on the system including the requirement to support precious and non-precious data and non-uniform communication within the physical architecture.

8.1.7.2 Description

In our architecture a number of requirements must be placed on the way data is organised.

A single global file system or name space will not scale given the non-uniform communication.

The dataflow approach leads to the existence of a large number of data objects within the system representing the input and output to the processing components. These data objects are distributed and linked through the system via the graph describing the overall dataflow. To implement such a system an object store is required.

To implement the concept of precious and non-precious data as well as manage the multi-threaded components spanning multiple compute nodes the object store must be capable of spanning a group of nodes. The presentation to the components could be as a parallel file system with a name space for each group of nodes. These considerations give rise to the concept of a data island.

For efficiency data islands should not extend beyond a compute island, however strictly linking a data island to a physical compute island does not allow for supporting efficiently different work flows. This leads to the concept of a compute island being partitioned into one or multiple data islands – the extreme case is a partitioning in which each node is a separate data island.

The data volume through the system is large and is the leading cost driver. The required data input from CSP must be ingested, but there are design considerations concerning the metadata associated with each data element. The design driver is that the metadata associated with each visibility should be kept to a minimum size.

8.1.7.3 Implications

These considerations lead to the following architectural decisions:

- Compute islands are partitionable into one or more data islands – a data island may be associated with one or a number of nodes within a compute island
- A single namespace, object store or parallel file system may span a data island providing the mechanism to implement the concepts of precious and non-precious data and support multi-threaded components spanning multiple nodes.
- At ingest only metadata with a high cadence is merged into the visibility data stream. Only a single weight is associated with each visibility datum. Where possible other metadata (e.g. u,v,w coordinates, visibility time stamps and channel frequency identification) are calculated dynamically from the higher cadence metadata associated with groups of visibilities.
- Objects are distributed and have a structure to support the dataflow model, together with the detailed requirements just identified. Access to data objects must support different access modes and the ability to distribute, collect and reorder data through the system.
- Low cadence metadata and state information through the system are presented to components via the dataflow system. The interface between the dataflow and the Telescope Model is managed by LMC.

8.1.8 SKA Regional Centres

8.1.8.1 Motivation

Much of the analysis of SKA data will be performed on systems that are not owned by the SKA organisation. It will be vital to implement mechanisms to manage the efficient movement of data to remote data centres and to make it available to researchers on different continents. It will be important to identify end-to-end network paths that can be engineered such that the wide area network (WAN) links leading from the host countries are used in the most effective way, not carrying data that is then dropped by edge networks. Also, there should be mechanisms to avoid moving the same products multiple times on these WAN links. Designating particular centres to lead the SKA processing efforts in a region will enable effective use of the networks and could provide additional services to SKA, such as acting as off-site backup and data redistribution locations. Having a common base set of user services at these centres will simplify moving work between sites, and the sharing of analysis tools between sites.

8.1.8.2 Description

The DELIV work package is working a set of tools and services to enable Regional Centres (RCs). These include tools to manage the efficient transfer of data on large round trip time (RTT) networks, to track the location of data products and to log access information for propriety data. Services are provided to replicate product catalogue information enabling fast access in all regions. Other services provide user interfaces to all types of SKA users. International Virtual Observatory Alliance (IVOA) toos provide a common view of data at all sites and tools to enable remote visualisation of astronomy data, to avoid needing to move large products to end user systems and to enable collaboration between geographically separated viewers are specified.

8.1.8.3 Justification

Enabling RCs will enhance the capabilities of the SKA overall. They will perform analysis that has to be done, but for which no funding has been allotted. They will also enable optimisation of international WAN links, something that has proved critical for CERN's data distribution. They will enable the management of the distribution of data, avoiding having the same products moved multiple times. In addition they can provide additional services such as off-site backup that would reduce the need for SKA funded storage.

8.1.8.4 Implications

The SKA will to consider what services should be assigned to RCs and what types of agreements will be needed to make best use of these. If reprocessing is required, then assigning this work to an RC would reduce the need to provide funded resources to perform this work. Also as noted above, if they are used for off-site backup, the amount of funded storage could be reduced.

8.2 Flow of data in the SDP architecture

The top-level architectural model for the flow of data in the SDP imaging data processing is shown in Figure 11. The processing of non-imaging data is similar. Briefly, the processing begins by merging the visibilities and the telescope manager metadata in the ingest layer. The RFI excision and baseline dependent averaging also done in this layer.

The data then enter a network switch which enables data re-ordering before the data are assigned to data islands. The processing the proceeds with the majority of the communications happening within data islands and some inter-island communications. The processing on island happens both in near-real time (for the observation currently being made and stored in the buffer) and in a batch mode (for observations already in the buffer). After the processing is complete the science-ready data are stored in the Archive, either for external distribution or potentially for averaging with further observations of the same field.

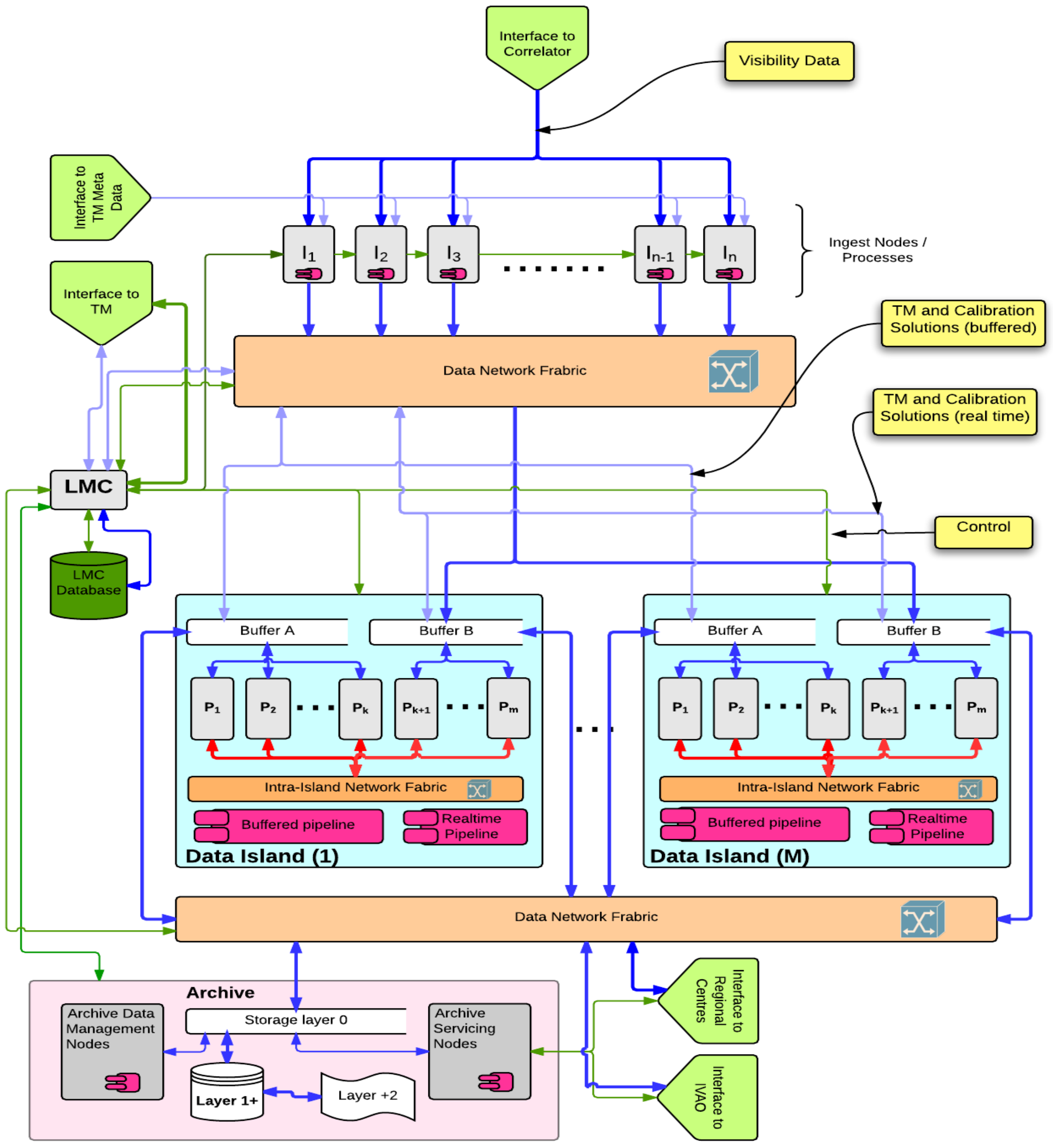

Figure 11: Illustration of the dataflow in the SDP System showing the concept of Data Islands (which have mechanisms for easy sharing of bulk data), the concept of Archive (where science-ready data products are stored) and the networking arrangements. The blue lines show the flow of high-volume data. The red-lines represent the low-latency communications. The green lines show the control and monitoring communication lines. The boxes are conceptual and do not correspond to specific product tree elements.

The flow of data and processing within the system are under the control of the Data Flow Manager and the data manager processes. The operation of the data flow system is discussed in RD30 with discussion of the data flow manager in RD 29 and details of implementation within the data layer in RD 30 and supporting documents. Here we summarise and introduce the main elements of our data flow architecture building on the discussions of the previous section.

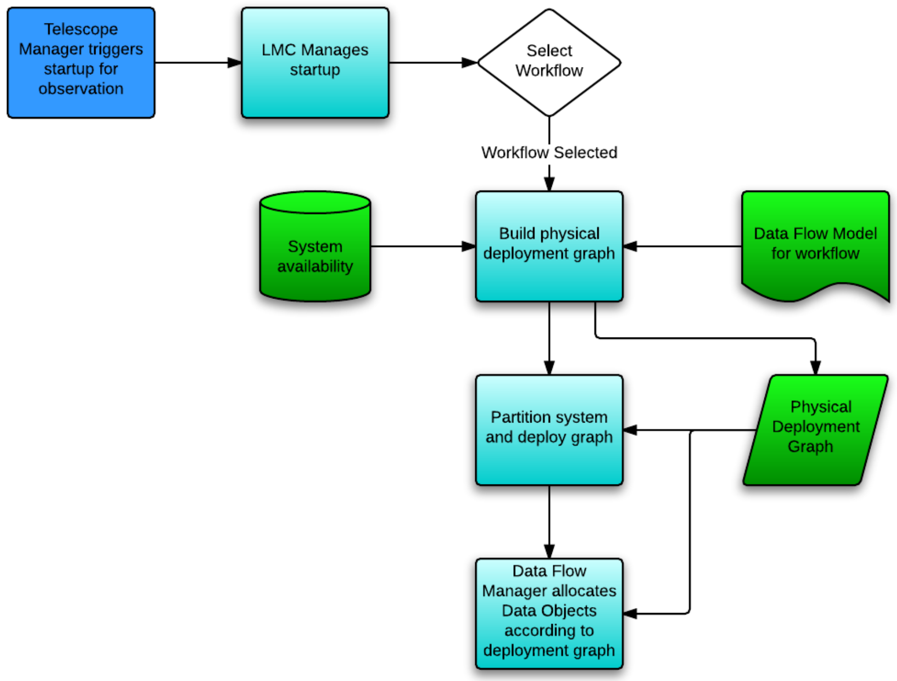
Telescope Manager triggers startup for observation
LMC Manages startup
Select Workflow
Workflow Selected
System availability
Build physical deployment graph
Data Flow Model for workflow
Partition system and deploy graph
Physical Deployment Graph
Data Flow Manager allocates Data Objects according to deployment graph

Figure 12: Illustration of the startup process within the context of the dataflow architecture

Figure 12 illustrates how an observation triggers the dataflow architecture. When a trigger is received from TM the SDP LMC Data Flow Manager component initiates the construction of a Physical Deployment Graph. This takes as input the required data flow model for the particular workflow as well as information on the state of the SDP system (node and other physical availability, scheduling constraints etc.). The physical deployment graph encapsulates all aspects of how the data-driven workflow is distributed across nodes in the system. The LMC initiates appropriate partitioning of the system, resource allocation and startup of local Data Manger processes. The Data Flow Manager initiates the distribution of the physical deployment graph and hence the creation of required data objects through the system – these objects are linked by the graph defining the data flow and also processing.

In Figure 13 we outline the flow of data through its lifecycle and interaction with the data manager processes.

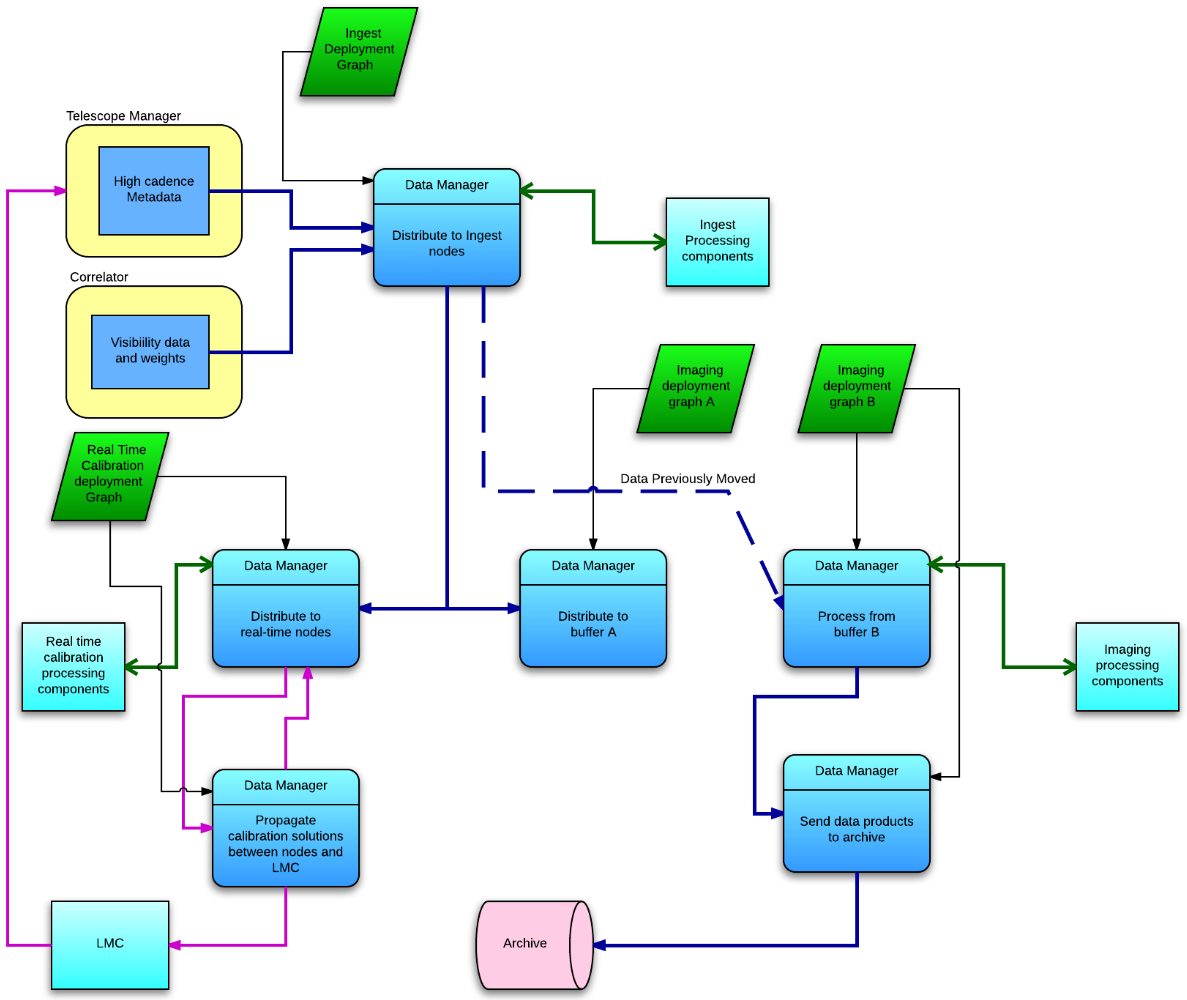
Ingest Deployment Graph
Telescope Manager
High cadence Metadata
Data Manager
Distribute to Ingest nodes
Ingest Processing components
Correlator
Visibility data and weights
Imaging deployment graph A
Imaging deployment graph B
Real Time Calibration deployment Graph
Data Previously Moved
Data Manager
Distribute to real-time nodes
Data Manager
Distribute to buffer A
Data Manager
Process from buffer B
Real time calibration processing components
Imaging processing components
Data Manager
Propagate calibration solutions between nodes and LMC
Data Manager
Send data products to archive
LMC
Archive

Figure 13: Illustrating the data lifecycle through the dataflow environment

Further details of this process are given in RD 30 and RD 26. The key aspects to note are that the data manger is responsible for managing the data flow and provides an interface to the various processing components in the system and all processing components are under the management of the data manager. Not only data product, but results of calculations, such as real-time calibration solutions are communicated via the dataflow environment

8.3 THE SDP SUB-SYSTEMS

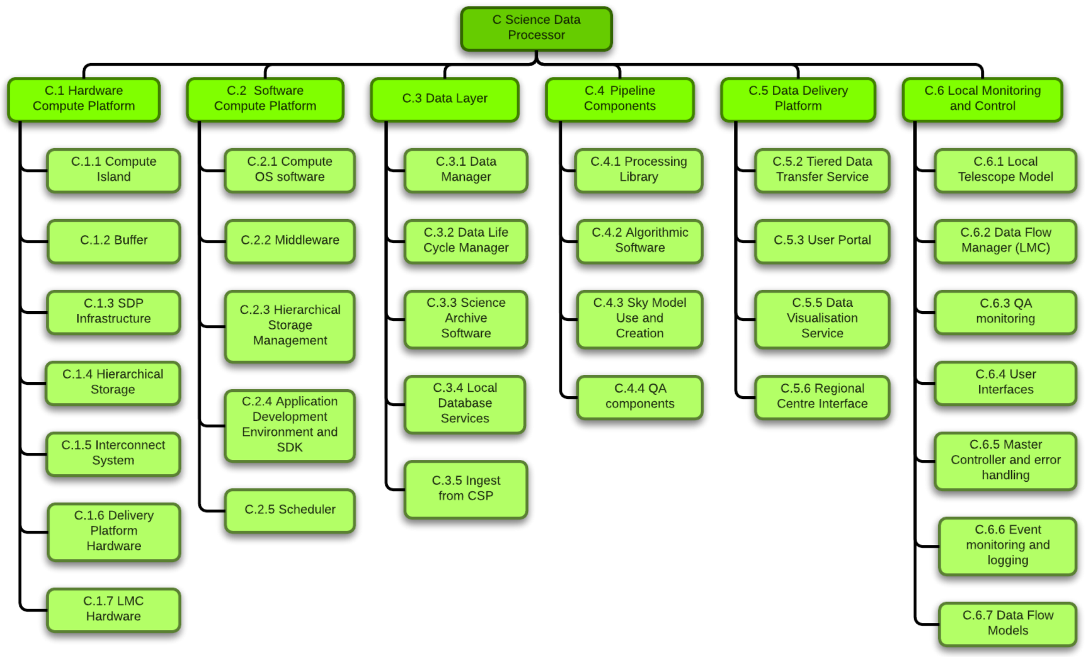
C Science Data Processor
C.1 Hardware Compute Platform
C.1.1 Compute Island
C.1.2 Buffer
C.1.3 SDP Infrastructure
C.1.4 Hierarchical Storage
C.1.5 Interconnect System
C.1.6 Delivery Platform Hardware
C.1.7 LMC Hardware
C.2 Software Compute Platform
C.2.1 Compute OS software
C.2.2 Middleware
C.2.3 Hierarchical Storage Management
C.2.4 Application Development Environment and SDK
C.2.5 Scheduler
C.3 Data Layer
C.3.1 Data Manager
C.3.2 Data Life Cycle Manager
C.3.3 Science Archive Software
C.3.4 Local Database Services
C.3.5 Ingest from CSP
C.4 Pipeline Components
C.4.1 Processing Library
C.4.2 Algorithmic Software
C.4.3 Sky Model Use and Creation
C.4.4 QA components
C.5 Data Delivery Platform
C.5.2 Tiered Data Transfer Service
C.5.3 User Portal
C.5.5 Data Visualisation Service
C.5.6 Regional Centre Interface
C.6 Local Monitoring and Control
C.6.1 Local Telescope Model
C.6.2 Data Flow Manager (LMC)
C.6.3 QA monitoring
C.6.4 User Interfaces
C.6.5 Master Controller and error handling
C.6.6 Event monitoring and logging
C.6.7 Data Flow Models

Figure 14: SDP Product tree

The Science Data Processor is decomposed into subsystem as shown in the product tree in Figure 14 and in the stack **Error! Reference source not found.** and described below:

1. The *Hardware Compute Platform* contains all of the hardware in the SDP, including computing units for all of the pipeline processing functions, ingest, archiving, delivery, LMC and all of the networking needed to connect these functions internally. This system is described in detail in [RD 25].
2. The *Software Compute Platform* contains all of the system software and much of third-party supplied, domain independent software. It provides the Operating System and Middleware, the application software development kits and the software to manage the hierarchical storage to be used in hardware. It therefore provides the software development environment. This system is described in detail in [RD 25].
3. The *Data Layer*, which provides the SDP-wide database systems, the data models, the data access and data communication for all processing tasks, and the SKA archive software. This system is described in detail in [RD 26].
4. The *Pipeline Components*, which consists of the most radio-astronomy specific software. This includes the actors that carry out all of the computation on the astronomical data, the software

to extract and use the sky models, and the base astronomical libraries (e.g., coordinate system software libraries) to support the above. This system is described in detail in [RD 27].

5. The *Delivery Platform* which implements the interfaces to the SKA Archive for the external users and interfaces for any regional archive centres. This system is described in detail in [RD 28].
6. The *Local Monitoring and Control* which coordinates all of the other elements, does the coarse-grained scheduling of the dataflow programs for processing the input data, monitors, etc. This subsystem is described in detail in [RD 29].

This decomposition is highly horizontally integrated with most of the SDP top-level functions crossing almost all of the top-level SDP sub-elements. The advantage of such decomposition is possibility in cost savings, optimal reuse of hardware and software between functions. Additionally, this allows the experts in each area to concentrate on those areas; and such product tree matches closely to the current work-breakdown structure of the SDP design project.

The disadvantages are broad inter-element interfaces which make the system overall less flexible, programmatic risk since all functions depend on almost all level-1 product tree elements and the possibility of inefficiency since the specialisation of sub-elements will not be as high as in a more vertically integrated design. Additionally, the breath of requirements and functions in each level-1 product makes their correct capture and validation more complex and risky.

8.5 Relationship between SDP Systems and top-level functions

The tables below show the roles that the SDP systems, i.e., the level 1 elements in the SDP product tree have with each level 1 function of the SDP as a system.

System	F.1 Continuum Imaging	F.2 Spectral line Imaging	F.3 Ingest Data
C.1 Hardware Compute Platform	Computing resources for data processing, buffer for storage of visibilities for iterative calibration, interconnect for parallelisation, data re-ordering and data reduction.	Computing resources for data processing, buffer for storage of visibilities for iterative calibration, interconnect for parallelisation, data re-ordering and data reduction.	Computing resources for RFI flagging and averaging, buffer for storage of visibilities after ingest, interconnect for receipt of data from CSP, for parallelisation & data re-ordering.
C.2 Software Compute Platform	Abstraction from hardware, programming environment, non-domain specific software for communications, storage, etc		
C.3 Data Layer	Domain specific data handling services, data models, data movement, databases, execution of processing components		Receipt of data from the CSP and initial formatting. Domain specific data handling services, data models, data movement, databases, execution of processing components.
C.4 Pipeline Components	Processing routines that compute on the bulk data		
C.5 Data Delivery Platform	N/A	N/A	N/A
C.6 Local Monitoring and Control	Overall control, computation of dataflow configuration, the dataflow program driving the function, local telescope model services		

Table 1: Assignment of L1 SDP functions to L1 elements of product tree (1/5)

System	F.4 Real-time Calibration	F.5 Drift Scan Imaging	F.6 Science Analysis
C.1 Hardware Compute Platform	Computing resources for data processing, interconnect for parallelisation, data re-ordering and data gathers and reductions.	Computing resources for data processing, buffer for storage of visibilities for iterative calibration, interconnect for parallelisation, data re-ordering and data reduction.	
C.2 Software Compute Platform	Abstraction from hardware, programming environment, non-domain specific software for communications, storage, etc		
C.3 Data Layer	Domain specific data handling services, data models, data movement, databases, execution of processing components		
C.4 Pipeline Components	Processing routines that compute on the bulk data		
C.5 Data Delivery Platform	N/A	N/A	N/A
C.6 Local Monitoring and Control	Overall control, computation of dataflow configuration, the dataflow program driving the function, local telescope model services		

Table 2: Assignment of L1 SDP functions to L1 elements of product tree (2/5)

System	F.7 Imaging transient search	F.8 Non-imaging transient processing	F.9 Pulsar Timing Post Processing
C.1 Hardware Compute Platform	Computing resources for data processing, interconnect for parallelisation, data re-ordering and data gathers and reductions.	Computing resources for data processing, interconnect for data distribution and gathers	
C.2 Software Compute Platform	Abstraction from hardware, programming environment, non-domain specific software for communications, storage, etc		
C.3 Data Layer	Domain specific data handling services, data models, data movement, databases, execution of processing components		
C.4 Pipeline Components	Processing routines that compute on the bulk data (including data from the non-imaging processor from the CSP)		
C.5 Data Delivery Platform	N/A	N/A	N/A
C.6 Local Monitoring and Control	Overall control, computation of dataflow configuration, the dataflow program driving the function, local telescope model services		

Table 3: Assignment of L1 SDP functions to L1 elements of product tree (3/5)

System	F.10 Pulsar Search Post Processing	F.11 Update Global Sky Model	F.12 Archiving
C.1 Hardware Compute Platform	Computing resources for data processing, interconnect for data distribution and gathers	Computing resources for data processing, interconnect for parallelisation, data re-ordering and data gathers and reductions	Hierarchical Storage hardware for the archive data, interconnect for receiving and exporting the data from the archive.
C.2 Software Compute Platform	Abstraction from hardware, programming environment, non-domain specific software for communications, storage, etc		Abstraction from hardware, programming environment, non-domain specific software for communications. Software to manage the hierarchical storage system.
C.3 Data Layer	Domain specific data handling services, data models, data movement, databases, execution of processing components		Handling of data movement in and out of the archive
C.4 Pipeline Components	Processing routines that compute on the bulk data		N/A
C.5 Data Delivery Platform	N/A	N/A	N/A
C.6 Local Monitoring and Control	Overall control, computation of dataflow configuration, the dataflow program driving the function, local telescope model services		Overall Control

Table 4: Assignment of L1 SDP functions to L1 elements of product tree (4/5)

System	F.13 Regional Centre Interface	F.14 SKA Archive User Interface	F.15 SDP LMC
C.1 Hardware Compute Platform	Hardware for computing, web-services and external interfaces, interconnect to SaDT for distribution of data		Hardware for computing, including high-availability hardware for critical services, interconnect for data distribution and gathering. Cold start control points.
C.2 Software Compute Platform	Abstraction from hardware, programming environment, non-domain specific software for communications, storage, web-service software		Abstraction from hardware, programming environment, non-domain specific software for communications, storage, abstractions for hardware monitoring.
C.3 Data Layer	Retrieval of data form the archive, archive search and discovery		Communication of QA data from pipelines, monitoring data from hardware
C.4 Pipeline Components	N/A	N/A	N/A
C.5 Data Delivery Platform	Domain specific software for interfaces with regional centres	Domain specific software for presenting user interfaces to astronomers	N/A
C.6 Local Monitoring and Control	Overall Control		All LMC specific software for control, aggregating QA data, drill down into monitoring.

Table 5: Assignment of L1 SDP functions to L1 elements of product tree (5/5)

8.6 Description of subsystems and their roles

The SDP Subsystems and their role is described in the table below.

SDP System	Sub-system	Description, Roles and Discussion
C.1 Hardware Compute Platform	C.1.1 Compute Islands	Compute Islands are the basic replicable units of hardware for the main processing functions of the SDP. They contain their own low-communication-cost internal interconnect and are further connected to other islands and other parts of the system via C.1.5 Interconnect System. 8.6.1.1.1.1.1.1 Compute Islands may be partitioned into a number of Data Islands. The roles of the compute islands are processing of all of the bulk data, including the ingest phase, imaging, calibration, pulsar processing and science analysis. Compute islands are a generalisations of a Map-Reduce concept of a rack as well as a widely used concept in modern HPC as a unit of deployment of a system.
	C.1.2 Buffer	The buffer sub-system implements the double-buffering of visibilities for imaging and calibration. The buffer can also be used to host working datasets from the archive, e.g., because they are to be averaged with further observations of the same field before being written back to the archive. The buffer is a key part of SDP architecture and also, due to the very high input data rates, an important cost driver and driver of design
	C1.3 SDP Infrastructure	This subsystem provides the SDP-provided infrastructure (e.g., racks) and interfaces with the local infrastructure elements for all other SDP infrastructure needs.
	C1.4 Hierarchical Storage	Provides the storage system for the science-ready data products, i.e., the SDP archive. Its receives the science-ready data from the Compute Islands and serves it to the C.5 Data Delivery platform and also back to the Compute Islands if reprocessing of data held in the archive is required
	C1.5 Interconnect System	Provides the connection to SDP external data interfaces, the inter-compute island communication, communication between Compute Islands, Hierarchical Storage, LMC Hardware and Delivery Platform Hardware. The interconnect system plays the critical role of conveying the CSP data into the SDP computing system and then carrying all of the inter-Compute Island communications for processing phases such as calibration, construction of continuum images and aggregation of spectral channels into spectral cubes.
	C1.6 Delivery Platform	Provides the computing hardware for provision of the data delivery functions. The design of this hardware will be different to the

	Hardware	Compute Islands because the function is substantially different.
	C.1.7 LMC Hardware	Provides the computing hardware for the centralised parts of the local monitoring and control functions. The design of this hardware is different from Compute Island because of the specific function as well as a more stringent high-availability requirements.
C.2 Software Compute Platform	C.2.1 Compute OS software	Provides the operating systems running on SDP hardware (C.1). The functions include local and distributed filesystems, lowest levels of networking stacks, memory and hardware accelerator management.
	C.2.2 Middleware	Provides the domain-independent software layers which build on top of the operating system functions and are used by the higher level layers. Functions like distributed object stores, high-performance messaging, events systems may be part of the middleware sub-system. Some of the critical roles of the middleware sub-system is providing efficient data movement and storage mechanisms for the Data Manger and the LMC systems.
	C.2.3 Hierarchical Storage Management software	Provides the software which manages and abstracts to operations of the C.1.4 Hierarchical Storage sub-system. It implements all of the domain-independent functions associated with the hierarchical storage including arranging migration of data between the storage tiers, tracking performance and maintenance of various tiers, internal redundancy and recovery and renewal of media.
	C.2.4 Application Development Environment and Software Development Kit	Provides the software tools for the development of all SDP-specific software. This includes compilers, interpreters, support libraries, software packaging tools, testing and profiling tools, code repositories and similar functions.
	C.2.5 Scheduler	Provides the software which allocates compute hardware for a particular capability, works around unstable or failed hardware, and limits the SDP hardware use to fall within the external limits, i.e., the electrical power and maximum temperature limits.
C.3 Data Layer	C.3.1 Data Manager	Provides the software for the full data abstraction layer for C.4 Processing Components and C.6 LMC. This includes the data models by which processing components access the data, the movement of data in the system and the dynamics of the interaction of processing components with their input and output data. The roles include handling the movement of data through the memory hierarchy down to storage within a Data Island and the inter-Data Island communications by message passing. The Data Manager also ensures the Processing Components are executed

		once their data are available.
	C.3.2 Data Life Cycle Manager	Provides the software that manages the movement, persistence and versioning of science-ready data products. Roles include ingesting the science-ready data products into the archive and reloading data from the archive back into the computing islands for further processing.
	C.3.3 Science Archive Software	Provides the software for the archive database and indexing capabilities. The key role of this is to provide the discovery and retrieval of data services necessary for the C.6 Data Delivery Platform to fulfil its functions of providing the interfaces to the astronomers and regional data centres
	C.3.4 Local Database Services	Provides the software for generic database services, to be used by higher level SDP sub-systems. Examples roles are database support to C.3.3 Science Archive Software, databases for source catalogues for component based sky models, C.6 LMC database requirements.
	C.3.5 Ingest Data from CSP into Data Layer	Provides the software to receive the data from the Central Signal Processor, to merge it with the Telescope Manager metadata stream and packetize it. The role of this sub-system is to for the first part of the ingest logical layer, before the data begins to be processed by the C.4.1 Processing Pipeline Ingest processing components such as Radio-Frequency Interference flagging.
C.4 Pipeline Components	C.4.1 Processing Library	Provides the software for all of the basic building blocks of the SDP pipelines. The components will be referentially transparent and of a fine enough granularity to allow a high level of parallelisation required. The components will present interfaces which are usable in a dataflow environment (although that will not preclude their use in an imperative programming environment). So that components can be deployed and re-used anywhere within the data-flow system these components must be stateless. The role of the processing library is to perform all of the computational steps required for the science-related SDP functions (ingest of data, spectral line and continuum imaging, detection of slow transients, real-time calibration, confirmation of pulsar candidates, confirmation of transients in time-series data, timing of pulsars). The processing library will not handle data movement, communications and similar functions which shall be provided by C.3.1 Data Manger.
	C.4.2 Algorithmic Software	Provides numerical and algorithmic software libraries which are used by one or more Processing Components but which present a conventional interface, e.g., a C-like interface, rather than a dataflow interface. This may for example include FFT libraries, libraries for world-coordinate conversions and calculations, atmospheric refraction libraries and similar.

	C.4.3 Sky Model Use and Creation	Provides the software libraries for suitable models of the sky that are compact, and can be efficiently used for source subtraction and calibration. The data representation of the sky model will be distributed by C.3 Data Layer but the creation of this representation and then using it later is handled by this sub-system.
	C.4.4 QA components	Provides the software processing components for producing quality assurance summaries from the observed data, science-ready results and intermediate pipeline processing stages, and for calculating the quantitative performance metrics. Example role would be components to produce RMS noise estimates from residual images as performance metric of the scientific quality of the observation.
C.5 Data Delivery Platform	C.5.2 Tiered Data Transfer Service	Provides the software for moving science-ready data to remote sites, e.g., Regional Centres. The role of this software is to allow efficient data distribution and scientific analysis, and to allow the use of regional science centres as a distributed mirror archive of the SDP archive (an efficient mechanism to fulfil SKA1-SYS_REQ-2350).
	C.5.3 User Portal	Provides the software for a web-based platform that will be the interactive interface for all users of the SDP archives. Typical role is providing an interface for users to search the content of the archive for particular observations and retrieve them.
	C.5.4 Data Discovery Service	Provides the software for data-discovery services to astronomers, e.g., by querying via IVOA standard interfaces
	C.5.5 Data Visualisation Service	Provides the software for visualising the data in the SDP archive without having to transfer the data themselves. As the volume of data in the SKA archive is likely to be so high this is likely to be efficient way of visualising data in most situations.
	C.5.6 Regional Centre Interface	Provides the software to interact with SKA regional centres, which may provide additional tiers of data distribution as well as computational resources for further scientific analysis of SKA data.
C.6 Local Monitoring and Control	C.6.1 Local Telescope Model	Provides the software that will provide an abstraction to the Telescope State information received from the Telescope Manager, the calibration parameters calculated by the SDP and the global and local sky models. The role of this software will be to ensure efficient and timely distribution of this information to all compute nodes and coherent system wide update for the sub-elements of the model that change (e.g., updating the sky model every major cycle).
	C.6.2 Dataflow Manager	Software to construct a static partitioning of the graph of data flows and data dependencies between the pipeline components so that the bulk movement of data is minimised and message rates are kept to the minimum between compute islands.

	C.6.3 QA monitoring	Software to receive, post-process and locally interpret QA data received from the QA elements of the processing pipelines.
	C.6.4 User Interfaces	Software to present the QA data obtained by C.6.3 QA monitoring to the operators of the telescope. This will include a visual environment for quickly overviewing the data and will allow users to interact with the calculation of the metrics.
	C.6.5 Master Controller and Error Handling	Software to the centralised control of the entire SDP element and the main contact point for the Telescope Manager. Controls and instantiates SDP capabilities, interacts with C.2.5 Scheduler to allocate computing resources to capabilities. Handles error conditions in the SDP and communicates them to the Telescope Manager.
	C.6.6 Event monitoring and logging	Software that does the SDP-element wide event monitoring and logging, provides for aggregation and drill-down functionality and communicates these data back to the Telescope Manager as requested.
	C.6.7 Data Flow Models	Software and configuration providing the top-level description of all of the pipelines and processing modes of the SDP. The models will specify how processing component sub-elements are connected together with data dependencies to implement all of the science and QA functions of the SDP.

Table 6: Descriptions of the SDP subsystems

8.7 Stack View of SDP sub-systems

A *stack* view of the SDP product tree elements is shown in Figure 15, where layers are drawn so that they can only use functions provided by products layers below them. This diagram includes every level 2 element of the SDP product tree, including hardware and software, to show the relationships in the complete system. This view illustrates the horizontal integration and reasonably good stratification in the decomposition: the layers corresponding to level one elements tend to be together. The two main exceptions are the LMC and Data layer which is understandable given their diverse role in the system. One of the key items of ongoing work is to understand how much of lower layer functionality will be fully abstracted away and how much will need to be exposed in some detail to the upper layers. This will be essential to fully understanding the risks and sequencing of the construction part of the SDP project.

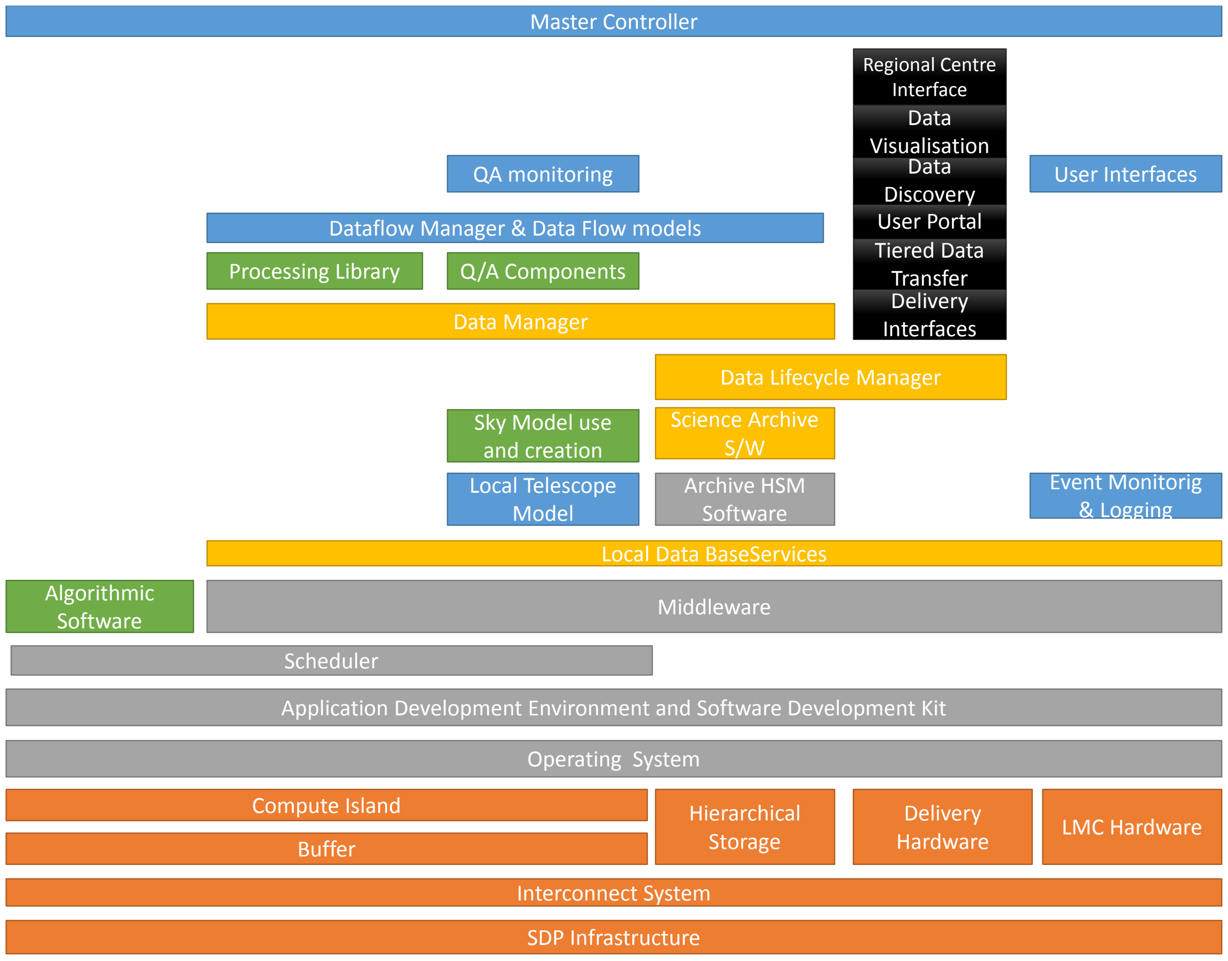

Figure 15: The Science Data Processer System stack, showing the relationship between the level 2 elements of the product tree. Boxes are arranged so that each box is allowed to use only the boxes below it. Furthermore, the horizontal partitioning of boxes into columns is approximately arranged so that boxes in vertical alignment tend to be used together. The boxes are colour coded to reflect the L1 elements of the product tree that they come from: Computer Hardware (orange) Computer Software (gray), Pipelines (green), Data (yellow), Deliver Platform (black), LMC (blue).

9 ARCHITECTURE DEVELOPMENT AND RISKS

In this section we describe the outline the plans for further development of the architecture and the currently perceived architectural risks.

The further development of the architecture will be focused on functions to elements of the product tree at deeper levels of both functional and product trees and then analysis and validation of requirements on these functions and the products that implement them.

A *relationship matrix* view of all elements of the SDP level-2 product elements is shown in Figure 16. This shows all pairs of L2 elements (belonging to different L1 elements) that have been identified to have a relationship, and all of these will need some further definition and verification. This relationship may take form of a requirement on one which is driven by the functions expected in the other L2 element, or it may be a formal interface definition, or be a more detailed description of their interaction at system build time, configuration time or run time. The on-going work on architecture will be to document further these relationships and to seek ways to simplify and diagonalise this relationship matrix in order to simplify the system.

9.1 CONSTRUCTION, INTEGRATION & COMMISSIONING RISK

The proposed architecture is not an incremental modification of architectures of existing systems for data analysis in radio astronomy but a significant break with past designs that will require significant amount of new design, construction of new software and commissioning. The size of the software construction project means it will not be a small team. The tools and techniques to be used will not be familiar to a large majority of scientific programmers.

Past experience in such projects is that unless very carefully and strictly managed and controlled there is a risk of time overruns, integration problems and periods of commissioning that are much longer than anticipated. There is a greater period of time between natural milestones in such non-incremental projects and unavoidably missing functionality during much of development. Tracking the progress of such projects is therefore much harder.

In addition, the way that the SKA project is currently structured lends itself to the formation of widely distributed consortia to perform the element level work. A widely distributed software team developing a complex and inter-related set of software components is known to be difficult and risky.

This risk can be reduced by:

- Strong organisational structure during the construction and commissioning of the SDP
- Maintaining a single, clear, focus on the delivery of well-performing software
- Careful planning of development stages and milestones so that progress can be tracked

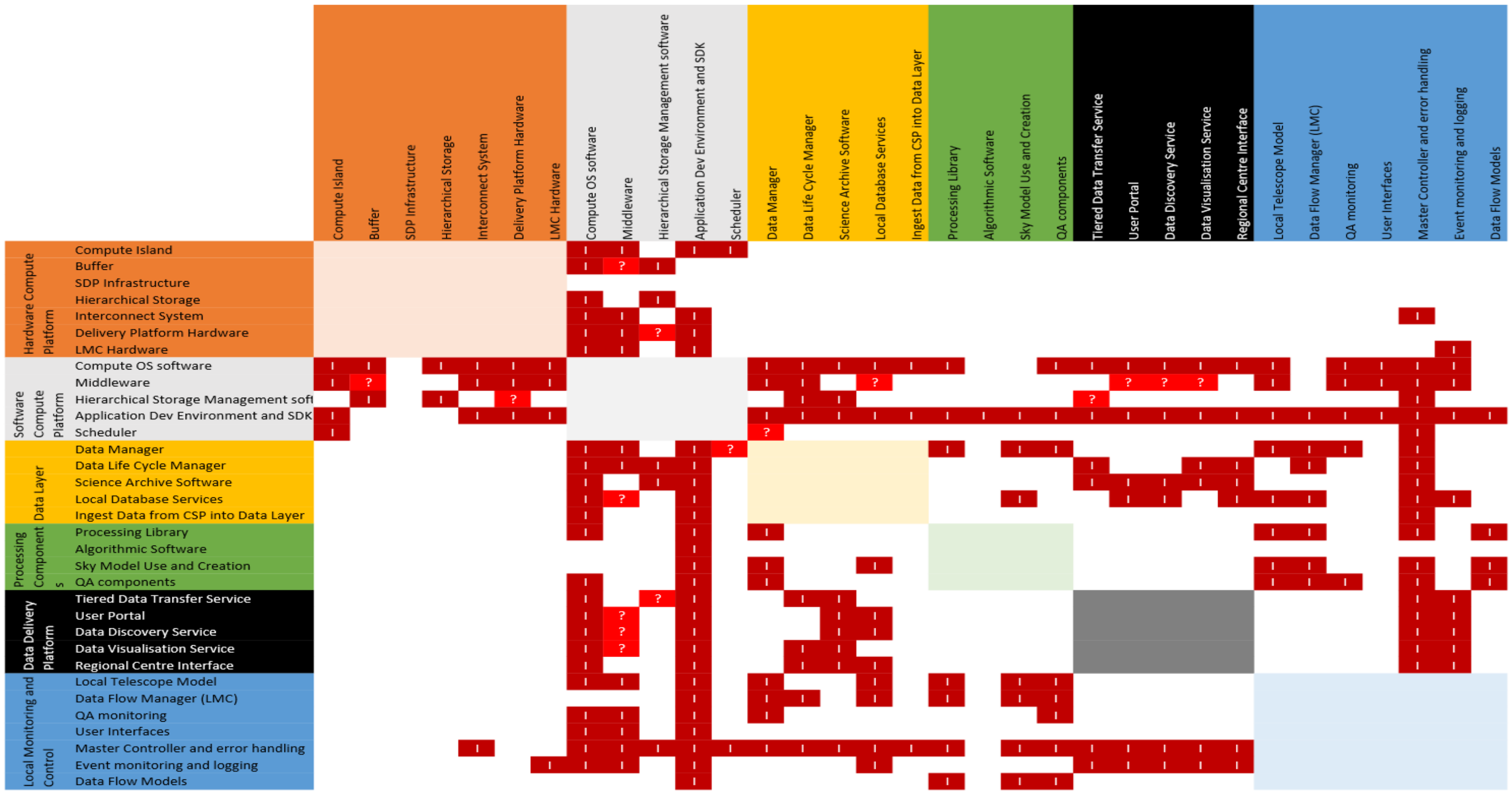

Figure 16: Matrix view of all level 2 elements of the SDP product tree showing (marked by I) where there are likely to be Interactions, interfaces or requirements that need to be established jointly. Areas where this may be the case are shown by ? and a lighter red colour. The rows and columns are colour coded according to the L1 element of the product tree that they belong to, using the same scheme as in ***Error! Reference source not found.*** *. Areas within each L1 element (coloured blocks on the diagonal) are not marked as this definition can be done within the L1 product element without needing cross element interactions.*

9.2 HORIZONTALLY INTEGRATED DESIGN

The decomposition of SDP is highly horizontally integrated, with most functions crossing most sub-systems. This presents potential risks in insufficient interface specifications (because interfaces are quite broad) and reduced flexibility in changing interfaces and specifications when the need arises (because subsystems are used by multiple functions placing multiple constraints on them). The programmatic risks of construction are somewhat increased in that delay to one subsystem will delay the delivery of almost all functions.

These risks can be reduced by early prototyping and integration, and a concerted and thorough documentation system of requirements and interfaces within the system engineering process.

9.3 NOVELTY/COMPLEXITY IN THE DATAFLOW SOFTWARE SUBSYSTEM

The dataflow system proposed has some relatively novel as well as some complex elements, e.g., the concept of non-precious data, the handling of large input datasets and the scheduling algorithm. The dataflow programming model as a whole and these features in particular are not supported by large software vendors in contrast how software like MPI is supported.

This risk can be reduced by prototyping and ongoing work to identify existing dataflow systems which have feature sets that are close to what is required.

9.4 SCALABILITY OF THE ARCHITECTURE

The architecture has been designed with the principle aim of scalability. Nevertheless, it is possible for some functions that it is not sufficiently scalable. This would most likely happen if certain stages of processing map very poorly to the dataflow programming model, e.g., because they require very small tokens flowing between many pairs of nodes. Alternatively, it could be the case that for some data processing requirements there are no dataflow program schedules which can reduce the communications between the compute islands sufficiently and consequently the networking systems are pushed to their limit.

This risk can be reduced once a full analysis of the dataflow representation of all of the major functions of SDP is done and these show a good mapping. Additionally, the introduced concept of major-cycle-synchronous global data store allows significant reduction in the small token rates.

SCIENCE DATA PROCESSOR

PDR.01 SDP Architecture

Document number	SKA-TEL-SDP-0000013
Context	ARCH
Revision	2
Author	Nikolic
Release Date	2015-02-09
Document Classification	Unrestricted
Status	Draft

Document No: SKA-TEL-SDP-0000002
Revision: 2
Release Date: 2015-02-09

Unrestricted
Author: Bojan Nikolic
Page 1 of 59

PDR01 version2.4 (pa)

EchoSign Document History — February 09, 2015

Created:	February 09, 2015
By:	Verity Allan (vla22@mrao.cam.ac.uk)
Status:	SIGNED
Transaction ID:	XJEHE3U6T3W4A4J

"PDR01 version2.4 (pa)" History

Document created by Verity Allan (vla22@mrao.cam.ac.uk)
February 09, 2015 - 3:31 PM GMT - IP address: 131.111.185.15

Document emailed to Bojan Nikolic (b.nikolic@mrao.cam.ac.uk) for signature
February 09, 2015 - 3:36 PM GMT

Document viewed by Bojan Nikolic (b.nikolic@mrao.cam.ac.uk)
February 09, 2015 - 3:39 PM GMT - IP address: 131.111.184.26

Document e-signed by Bojan Nikolic (b.nikolic@mrao.cam.ac.uk)
Signature Date: February 09, 2015 - 3:40 PM GMT - Time Source: server - IP address: 131.111.184.26

Document emailed to Paul Alexander (pa@mrao.cam.ac.uk) for signature
February 09, 2015 - 3:40 PM GMT

Document viewed by Paul Alexander (pa@mrao.cam.ac.uk)
February 09, 2015 - 6:48 PM GMT - IP address: 131.111.185.15

Document e-signed by Paul Alexander (pa@mrao.cam.ac.uk)
Signature Date: February 09, 2015 - 6:49 PM GMT - Time Source: server - IP address: 131.111.185.15

Signed document emailed to Verity Allan (vla22@mrao.cam.ac.uk), Paul Alexander (pa@mrao.cam.ac.uk) and Bojan Nikolic (b.nikolic@mrao.cam.ac.uk)
February 09, 2015 - 6:49 PM GMT