Near-real-time satellite data processing at NIWA with Cylc

CSPP/IMAPP Users' Group Meeting

SSEC, Madison, Wisconsin
June 27-29 2017

Simon Wood, NIWA

NIWA
Taihoro Nukurangi

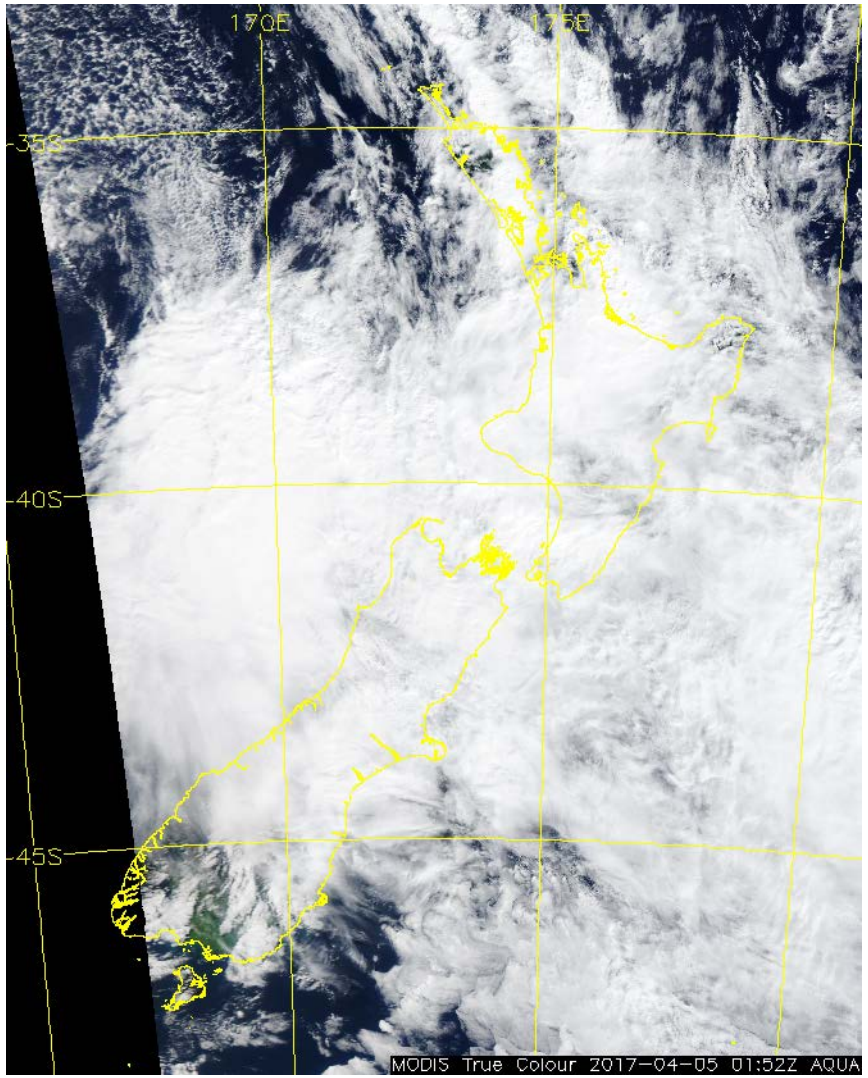
Outline

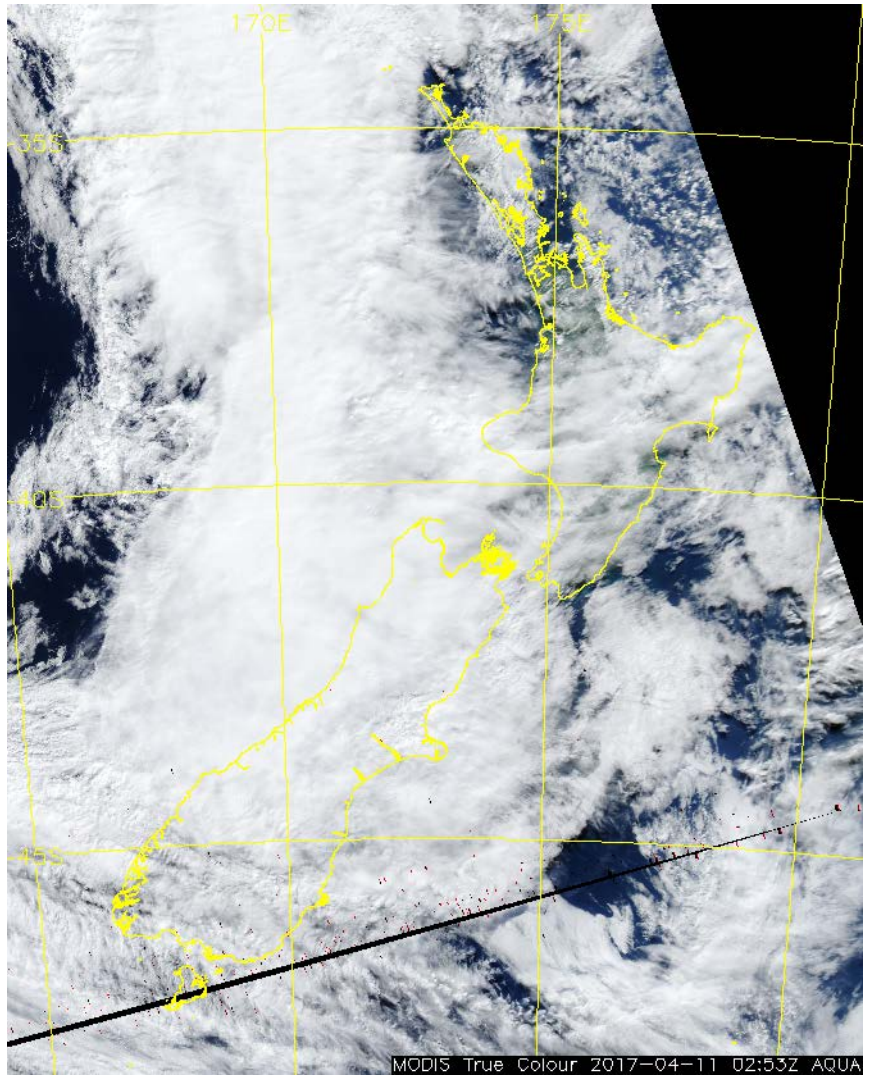
- About NIWA – who we are and what we do

- NIWA's Direct Readout and Data Processing facilities
  - What facilities do we have?
  - Brief history and future directions
- Intro to Cylc
  - What is Cylc?
  - Motivations and (very) quick tour
- Can we use Cylc to manage real-time satellite data processing?
  - Suitability
  - Some issues to address
  - Conclusion

NIWA
Taihoro Nukurangi

Aotearoa: ‘the land of the long white cloud’





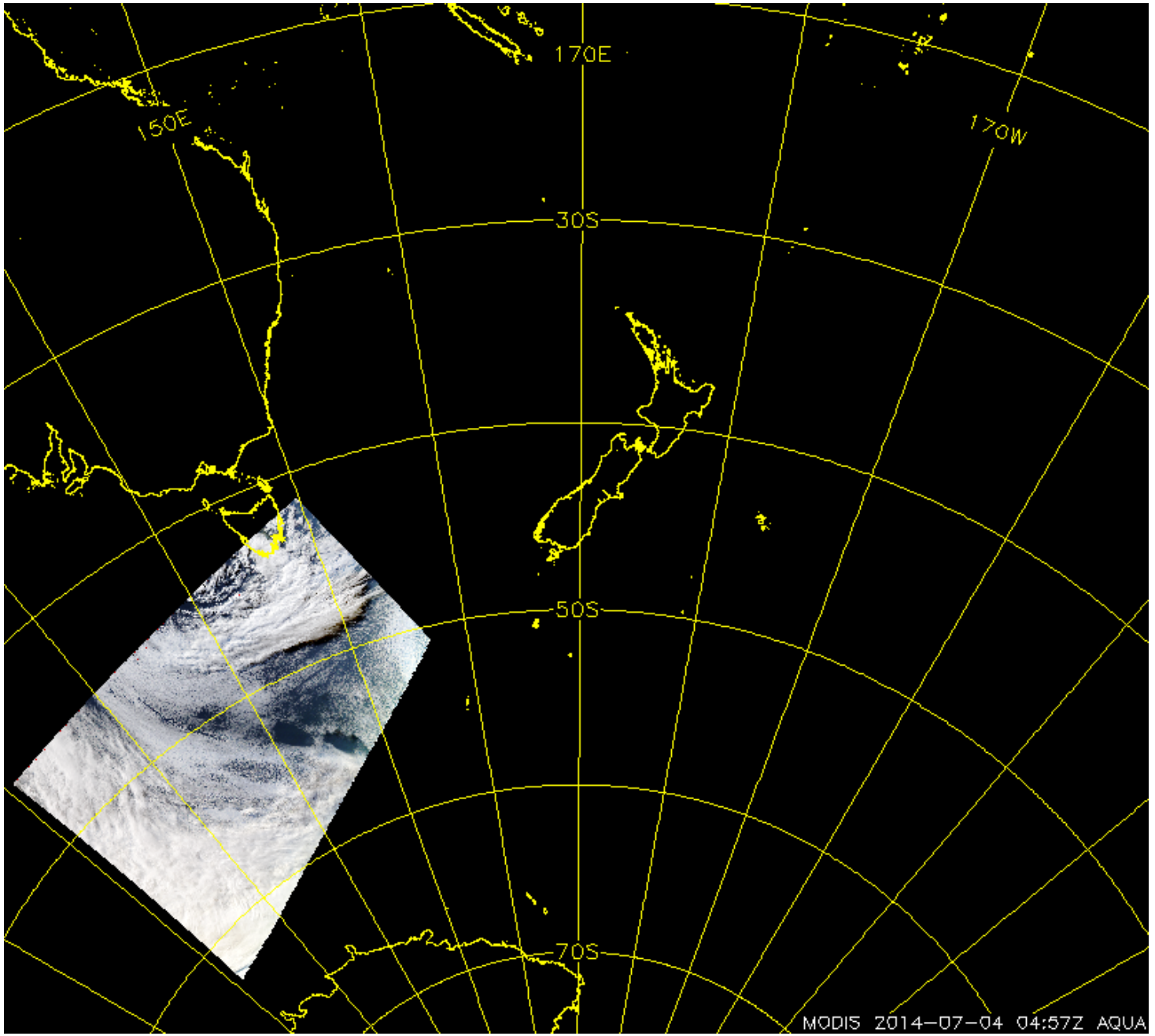
NIWA
Taihoro Nukurangi

But if we look from further out we see that we're just two small islands in a great big ocean

NIWA
Taihoro Nukurangi

Two small islands surrounded by ocean

- Remote sensing works really well over large expanses of ocean
- We need timely atmospheric observations for data assimilation into NWP
- So maybe we're really ideally placed?



NIWA
Taihoro Nukurangi

NIWA

Taihoro Nukurangi

Taihoro - flow and movement of water

Nukurangi - interface between sea and sky (atmosphere).

'Where the waters meet the sky'

- NZ Government Crown Research Institute (CRI)
- Purpose: to enhance the economic value and sustainable management of New Zealand's aquatic resources and environments, to *provide understanding of climate and the atmosphere* and *increase resilience to weather and climate hazards* to improve the safety and well being of New Zealanders.

NIWA is lead CRI for

aquatic resources and environments
oceans
freshwater and marine fisheries
aquaculture
climate and atmosphere
climate and weather hazards
aquatic and atmospheric-based energy resources
aquatic biodiversity and biosecurity

Science Centres

Climate and Atmosphere
Natural Hazards
Freshwater
Coasts and Oceans
Aquaculture
Fisheries

Other Centres

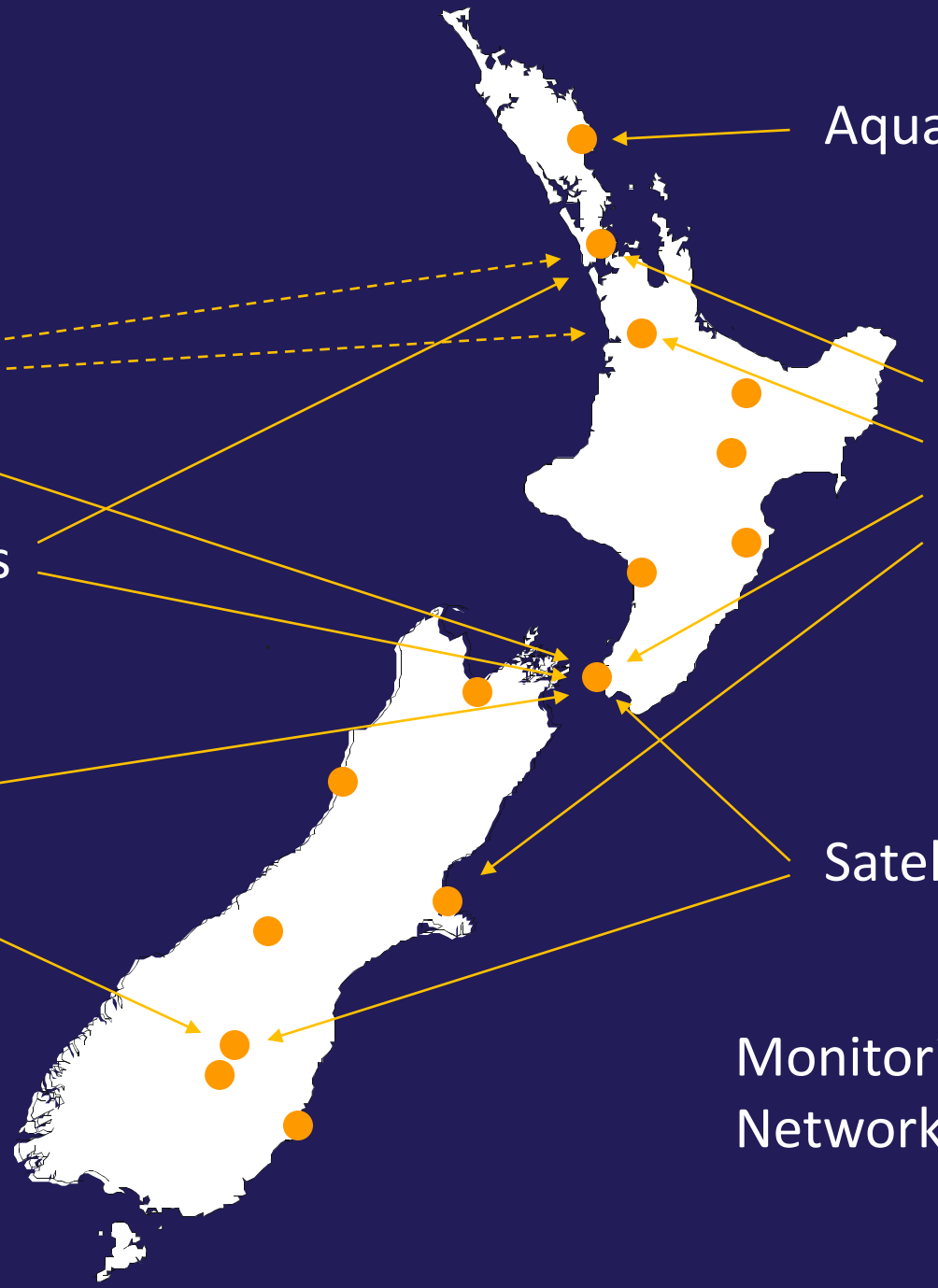
Environmental Information
Maori Development
Pacific Rim and International

NIWA
Taihoro Nukurangi

~630 Employees

Perth, Australia, 12

Key Facilities



Aquaculture Facilities

HPC

Collections & Databases

Atmospheric Observatories

Specialized Labs
- Mass Spec
- Water Quality
- Air Quality
- etc

Satellite Ground Stations

Monitoring Networks

NIWA
Taihoro Nukurangi

Key Facilities - Vessels

PP

NIWA
Taihoro Nukurangi

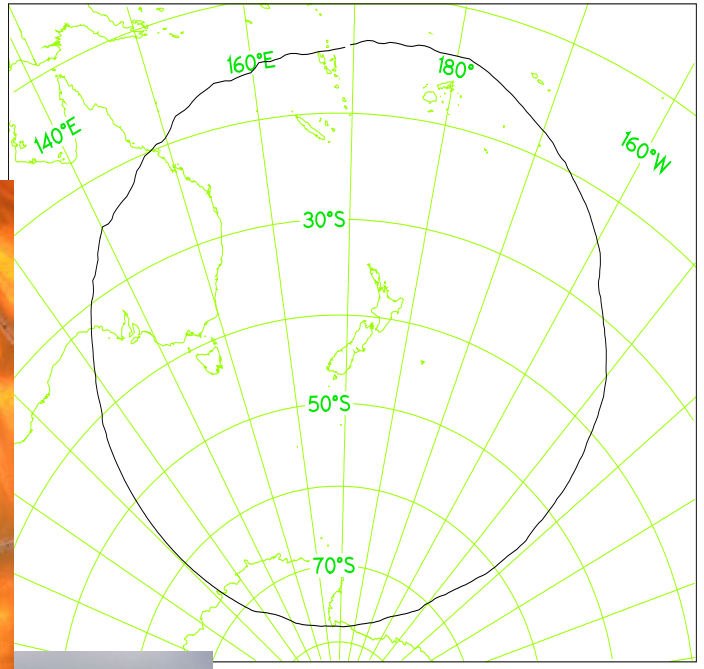
Satellite Reception Facilities 1: Maupuia

- First receiver installed in Wellington early 90s
- 1.2 m dish, L-band only
- Supplied by ES&S
- NOAA [15,] 18, 19
- L0 data back to Wellington Greta Point campus via radio link across the water (quite slow)
- Lovingly decorated by the locals…

NIWA
Taihoro Nukurangi

Satellite Reception Facilities 2: Lauder

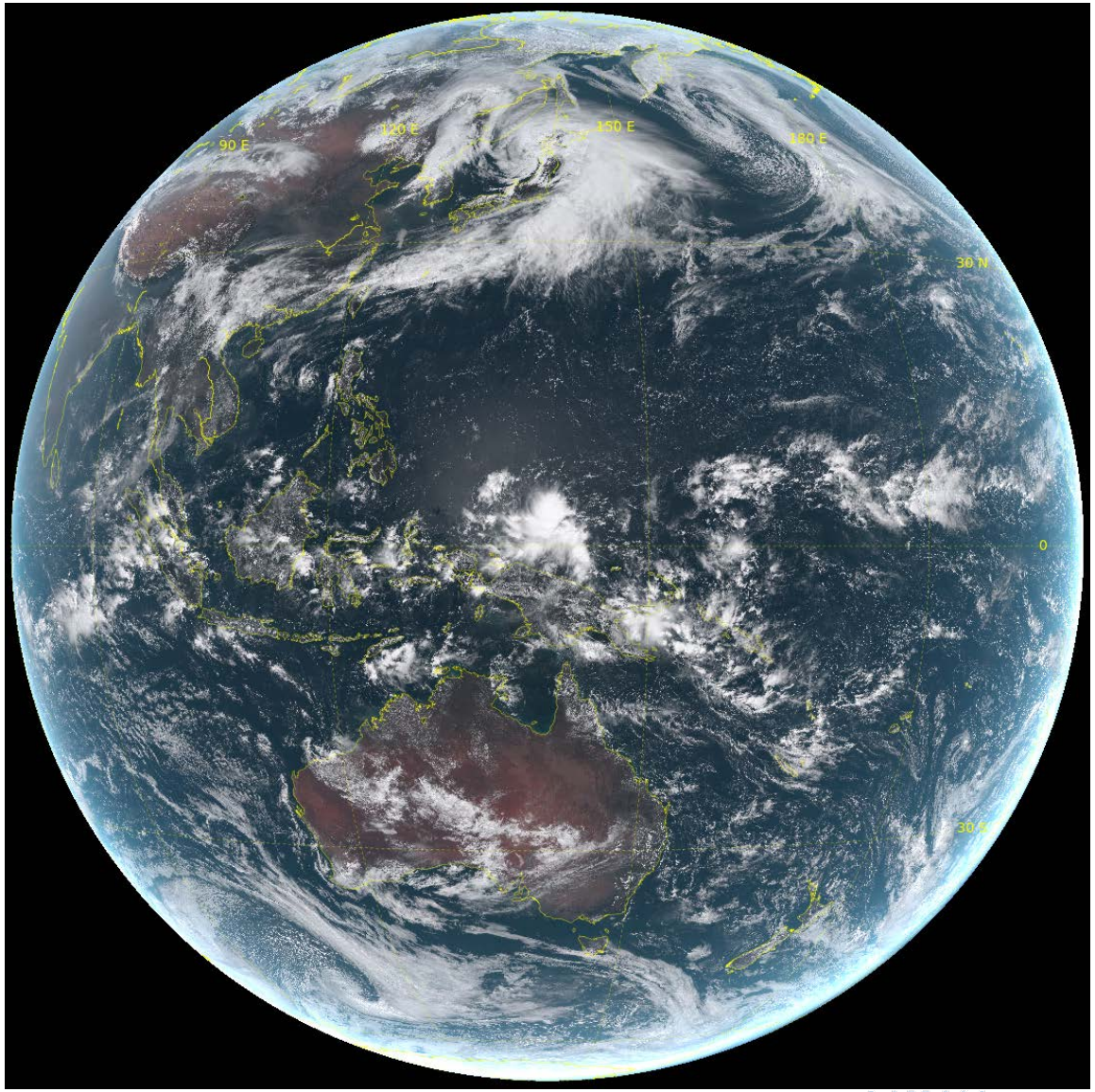
- ES&S, dual X-L band, 2.4 m antenna, A-B tracking mount
- Installed late 2007
- Lauder location chosen for (reasonably) good skyline and radio quietness
- NOAA, Metop, Terra, Aqua, NPP (FY also possible)
- L0 data back to Wellington via REANNZ (NZ's NREN) but 'last mile' can be problematic



NIWA
Taihoro Nukurangi

Satellite Reception Facilities 3: Himawari-8/9

- (Not Direct Readout, but…)
- NIWA obtains Himawari AHI imager data from the ‘HimawariCloud’ service (via the NZ MetService)
- AHI imager, all bands, full resolution
- Complete Earth disk scan every 10 mins
- All data at NIWA within 10 mins of end of scan
- Immediate processing to netCDF. netCDF files available within 10 mins of last segment file arriving at NIWA.
- Top of atmosphere image products routinely generated



NIWA
Taihoro Nukurangi

High Performance Computing Facilities – past and present

- 1999: Cray T3E 1200 (2nd hand ex UK Met Office)
  - Acquired specifically for the purpose of running localized versions of the UK MO's Unified Model (NWP)
- 2009, 2013: IBM P575/P6 supercomputer
  - Initially 58 nodes x 32 cores (1856 cores) @ 4.7 GHz, upgraded to 106 nodes (3392 cores) in 2013.
  - 64 or 128 GB RAM per node (28 have 128 GB)
  - 4 x 144 port InfiniBand switches, 4 more added in 2013
  - Originally 0.5 PB high speed disks with ~8 PB tape storage (via HSM), disks now 1.5 PB
  - GPFS and TSM (HSM)
  - 8-10 P520/P6 servers for management and login nodes
  - A BladeCenter with 56 x Xeon 2.53 GHz with 224 GB RAM for pre and post-processing tasks. Upgraded to 113 cores with 448 GB RAM in 2013.
  - A PureFlex system was added in 2013 with 60 x Xeon, 1 x Tesla K40 GPGPU and 1.4 TB RAM for pre and post-processing tasks.
  - Housed at NIWA Wellington (Greta Point) in purpose built computer room.
  - Owned jointly between NIWA (majority) and other research institutes under the NeSI (New Zealand eScience Infrastructure) umbrella.

High Performance Computing Facility – future

- Procurement for a replacement machine began mid 2016
- Contracts signed and public announcement mid June 2017
- Full details still being kept under wraps, sneak preview:
  - 3 machines (replacing both the IBM P575 and a 6000+ core x86 cluster at Auckland University)
    - 2 x Cray XC50 (one large capability machine, one much smaller for disaster recovery): x86 Skylake, approx 19500 cores (main machine), Infiniband EDR 100 Gb/s, Linux OS.
    - 1 x Cray CS400 capacity machine for pre- and post-processing or loosely coupled ('embarrassingly parallel') workflows: x86 Broadwell, 9216 cores, Linux
  - IBM Spectrum Scale (GPFS), 9.8 PB disk, 20 PB tape
  - Both large machines located at NIWA Wellington. DR machine in Auckland.
  - Joint ownership through NeSI.

Satellite Data Processing – beginnings and recent past

Various systems over the years:

- 1990s: Batch processing on VMS. Most processing code developed in-house.
- Early 2000s: Collection of cron (and similar) based scripts on Linux. Begin adoption of community software (e.g. AAPP, SeaDAS).
- 2008 onwards: SDPS (aka ‘satproc’) – dedicated, lightweight processing framework written in Python using ‘inotify’ file system monitor to detect arrival / generation of new input files and trigger appropriate processing task. Publish / Subscribe model.

  Worked well for last few years but now time to move on…

Satellite Data Processing – motivations for change

Why do we need a new system?

1. Processing environment is changing
   - Until recently we had dedicated machines for satellite processing
   - Now all satellite processing is expected to run in HPC environment alongside other operational workflows
   - inotify does not scale well to GPFS / cluster environments (blind to non-local filesystem events)

2. Processing culture is changing
   - Originally ‘operational’ processing was managed by science code ‘owners’ – satellite processing was managed by me (log tails via ssh on my desktop etc)
   - Now we have a 3-person Operations Team responsible for keeping all forecasting models and associated processing running 24x7. Satellite data processing will also become their responsibility. They don’t thing log tails are quite sufficient…

Operations Team Requirements

- Have to manage whole suite of forecasting models, observation processing and (image) product generation systems
- Need to be able to see status across many systems in a consistent fashion
- Manage ‘by exception’; don’t want to have to check that processing completed successfully – just be alerted when things fail
- Not domain experts – generalists
- Not programmers – but can do some basic scripting
- Prefer graphical interfaces
  - View status across whole ‘suite’
  - Start / stop/ restart / retry processing tasks easily (e.g. following intervention)
- Our ops team likes Cylc…

Is Cylc a good fit for satellite data processing?

Pros

- Designed to manage continuously repeating workflows
- Simple configuration via text file can describe arbitrarily complex workflows
- Support for HPC job schedulers
- Support for real-time processing
- Rich graphical monitoring tools
- Comprehensive logging
- Easy integration with alerting systems (e.g. Nagios)
- Already well established in our operations team; familiar interface

Cons

- Designed for *time based* cycling: “run over period from Jan 1 to March 31 in 6 hour steps starting at 00:00”
- Real-time support assumes *time based* cycling: “run every 3 hours starting at 00:00”
- Designed for continuous *cycling* workflows – implicit assumption of inter-cycle dependency (what happens when processing of previous pass fails?)

Issues 1, 2: time based cycling

Obviously real-time satellite processing cannot be scheduled on a “run every N minutes” basis; we need to start processing as soon as the data arrives (at irregular intervals through the day)

- Use Cylc’s *integer cycling* mode to remove any notion of time dependence:

```
[scheduling]
    cycling mode = integer
    [[dependencies]]
        [[[R1]]]        # first cycle; do some inits
            graph = prep => get_data
        [[[P1]]]        # run every cycle
            graph = get_data => proc1 => proc2 => products
```

- with *external triggering* to force each cycle to wait for an external trigger before starting:

```
[scheduling]
    cycling mode = integer
    [[special tasks]]
        external-trigger = get_data(scheduled pass end time reached)
    [[dependencies]]
        ...
```

Issues 1, 2: time based cycling (cont'd)

- We'll arrange for an external system to send a trigger at the end of each pass

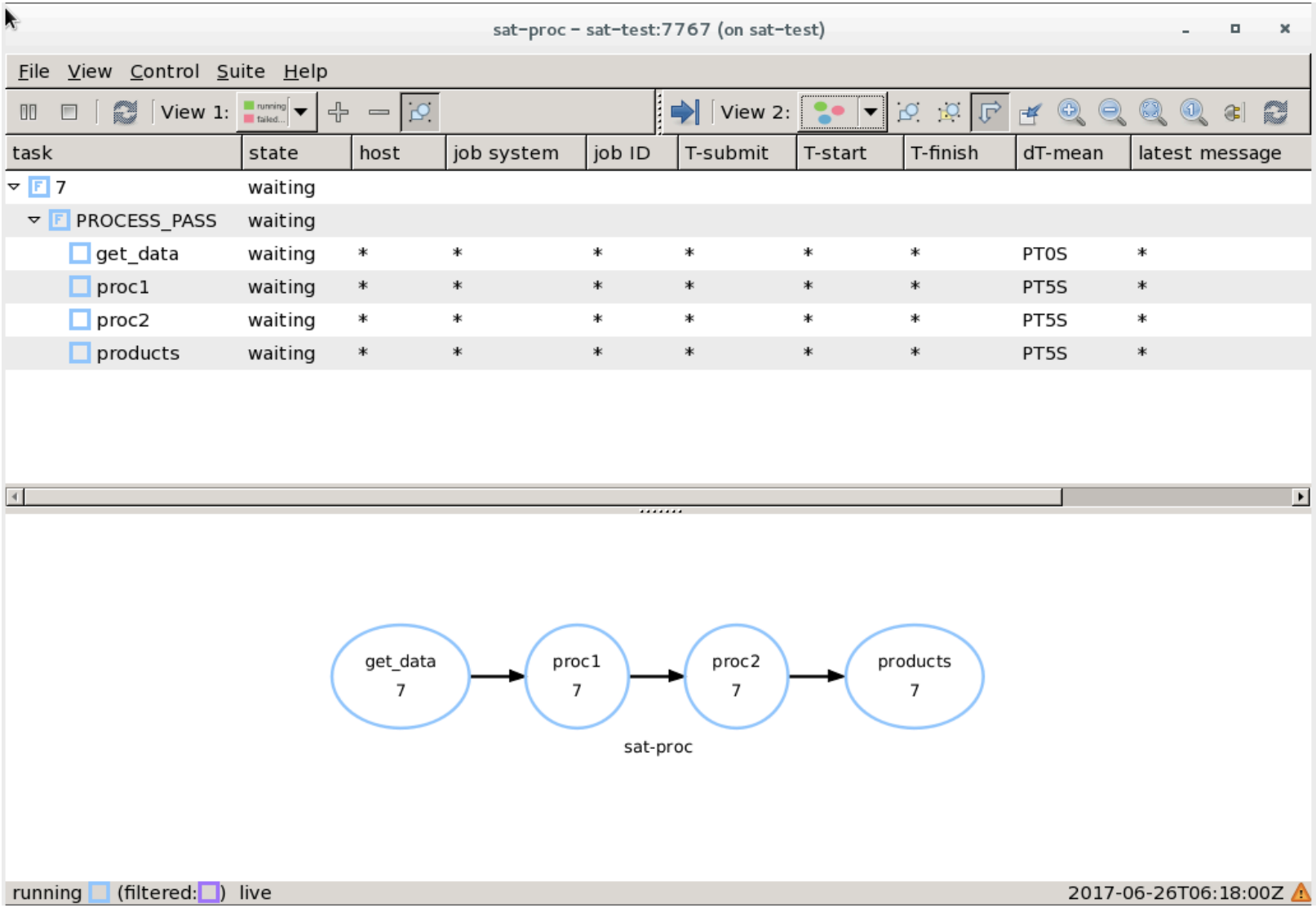
```
$ cylc ext-trigger <SUITE NAME> \
  "scheduled pass end time reached" <PASS_ID>
```

- this could be a separate time cycling suite that runs once a day to get the schedule file from the receiver and configures a timer chain to generate the trigger events at the required times, e.g.:

```
[scheduling]
    [[special tasks]]
        clock-trigger = get_schedule    # cannot run until due time
    [[dependencies]]
        [[[R1]]]
            graph = prep => get_schedule
        [[[PD1]]]                       # run every day
            graph = get_schedule => parse_schedule => queue_triggers
```

Issue 3: continuous cycling – normal operation

Each task spawns its own successor when it becomes ready to run (submitted state)



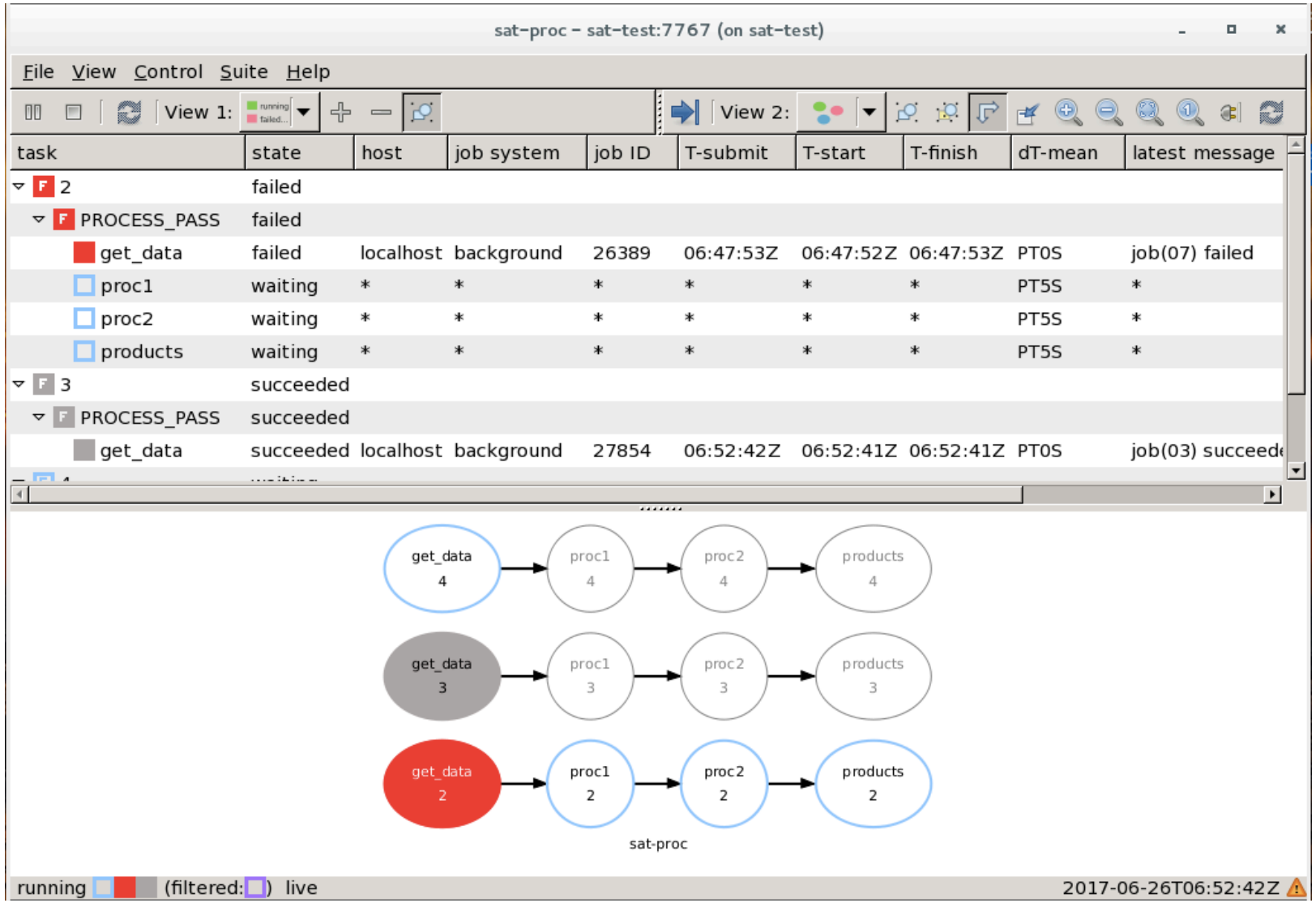
NIWA Taihoro Nukurangi

Issue 3: continuous cycling – example 1: no data

1. Previous cycle OK

2. *get_data* fails – no successors created for *proc1*, *proc2* or *products*

3. Suite stalls – nothing will get processed for sebsequent passes



NIWA Taihoro Nukurangi

Issue 3: continuous cycling – example 2: a processing task fails

1. Previous cycle OK
2. *proc2* fails – no successor created for *products*
3. Suite stalls – subsequent passes cannot complete

NIWA Taihoro Nukurangi

Issues 3: continuous cycling (cont'd)

Clearly we need to force creation of successors for the failed task's dependants.

- We can force any task to spawn it's own successors via the CLI:

```
$ cylc spawn <SUITE NAME> <TASK>.<CYCLE_POINT>
```

- We can associate an event handler to a task's failed state which causes a custom script to be run whenever the task enters the failed state. Here we need to get it to call `cylc spawn` on all dependants at the *current cycle point*:

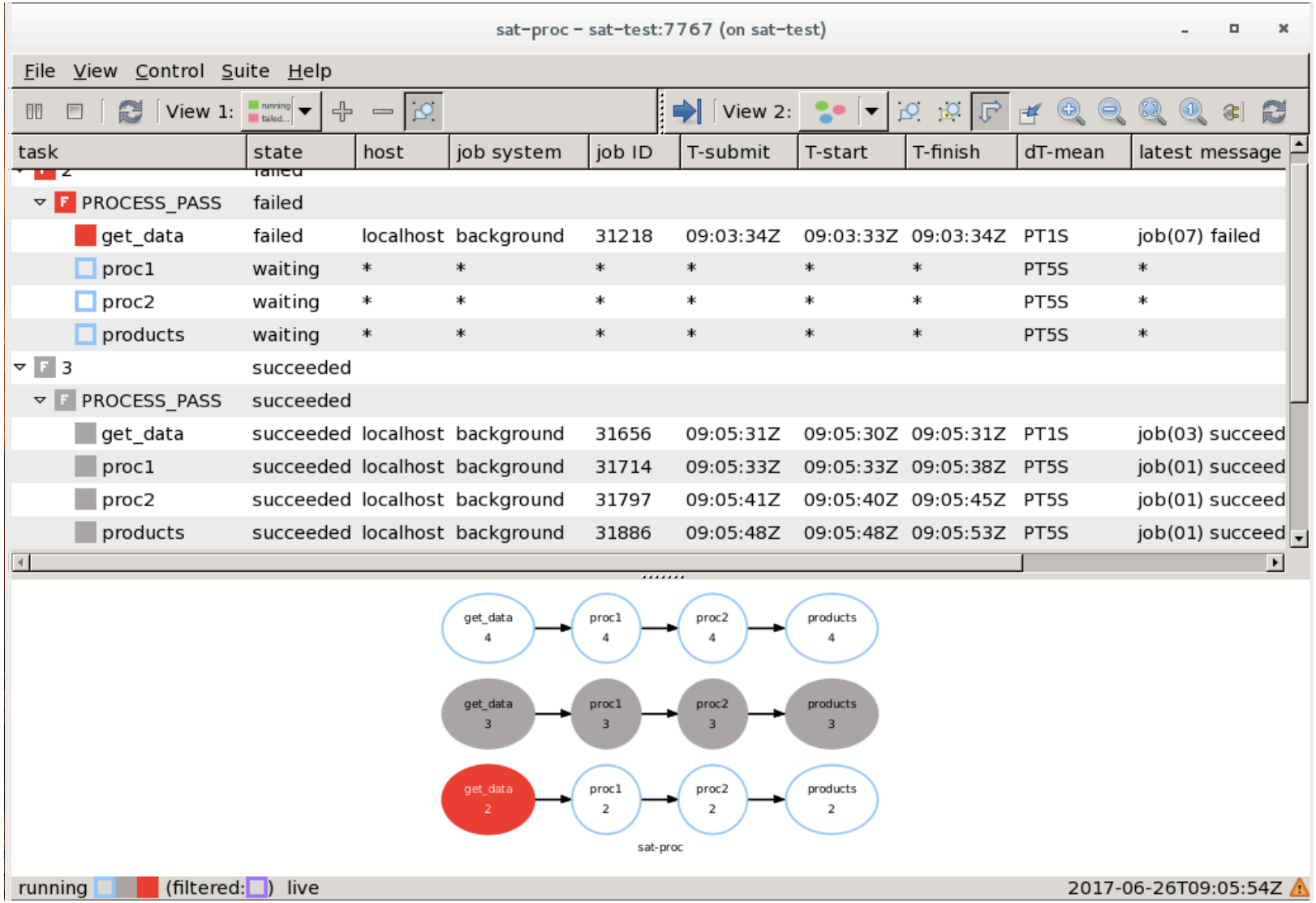
```
[runtime]
    [[get_data, proc1, proc2]]
        [[[events]]]
            failed handle = cylc spawn %(suite)s \
                                       PROCESSORS.%(point)s
```

where `PROCESSORS` is a task group containing `proc1`, `proc2` and `products`

Issue 3: continuous cycling – example 1 resolved

1. *get_data* fails as before but now all dependants' successors get created

2. Later passes can be processed normally

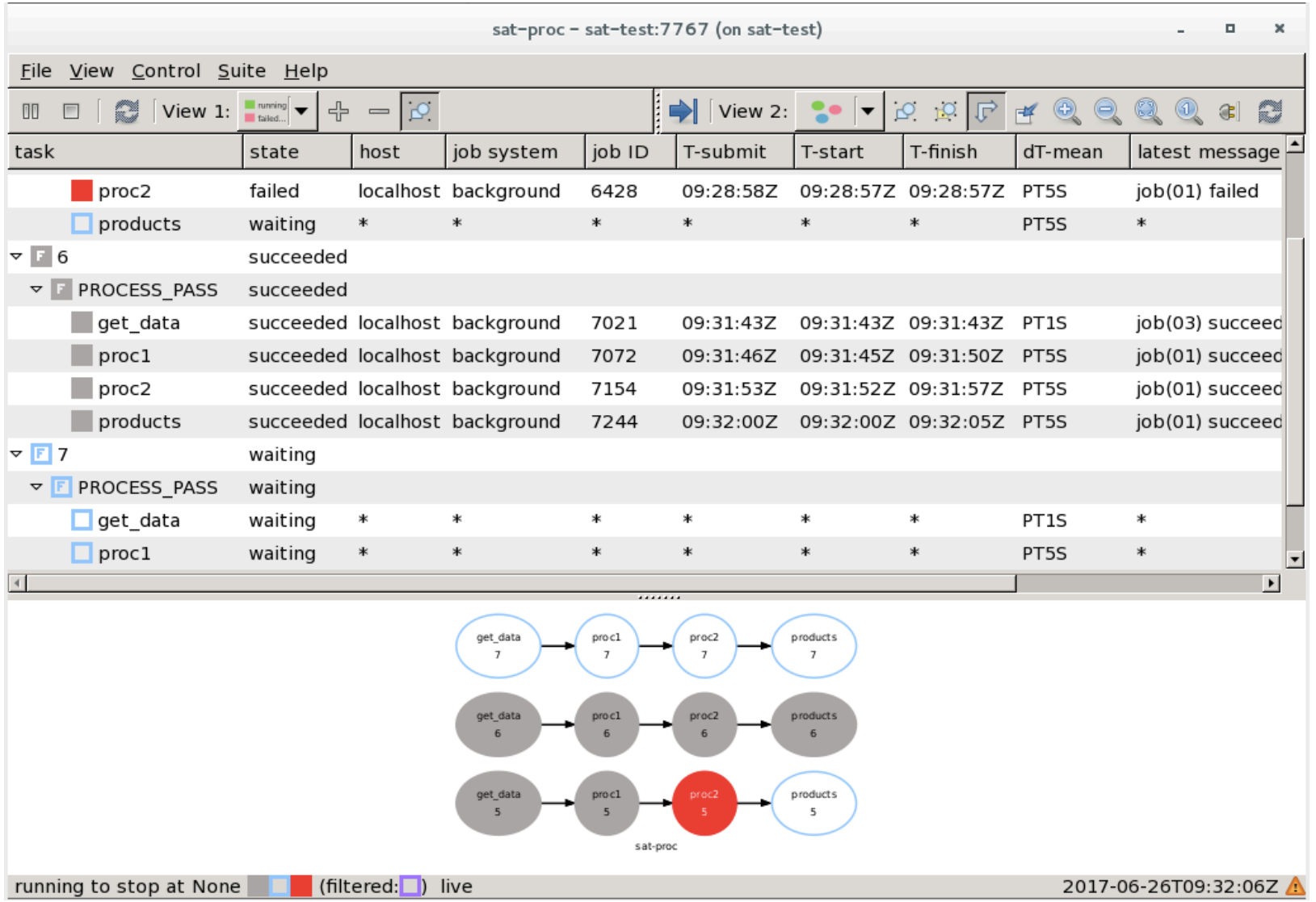


NIWA Taihoro Nukurangi

Issue 3: continuous cycling – example 2 resolved

1. *proc2* fails as before but now a successor gets created for *products*

2. Later passes can be completed normally



NIWA Taihoro Nukurangi

Is Cylc a good fit for satellite data processing?

Pros

- Designed to manage continuously repeating workflows
- Simple configuration via text file can describe arbitrarily complex workflows
- Support for real-time processing
- Rich graphical monitoring tools
- Comprehensive logging
- Easy integration with alerting systems (e.g. Nagios)
- Already well established in our operations team; familiar interface

Cons

- Designed for *time based* cycling: "run over period from Jan 1 to March 31 in 6 hour steps starting at 00:00" fixed with integer cycling
- Real-time support assumes *time based* cycling: "run every 3 hours starting at 00:00" fixed with integer cycling and external triggers
- Designed for continuous *cycling* workflows – implicit assumption of inter-cycle dependency fixed with `cylc spawn` from error handler