The LCOGT Network Scheduler

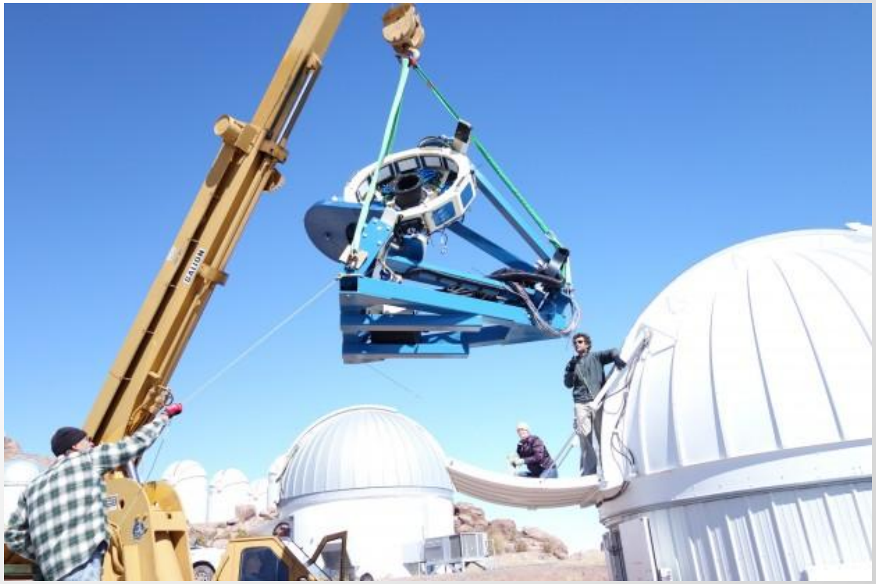
Lessons we learnt the hard way

Eric Saunders
Software Team Lead
Las Cumbres Observatory Global Telescope

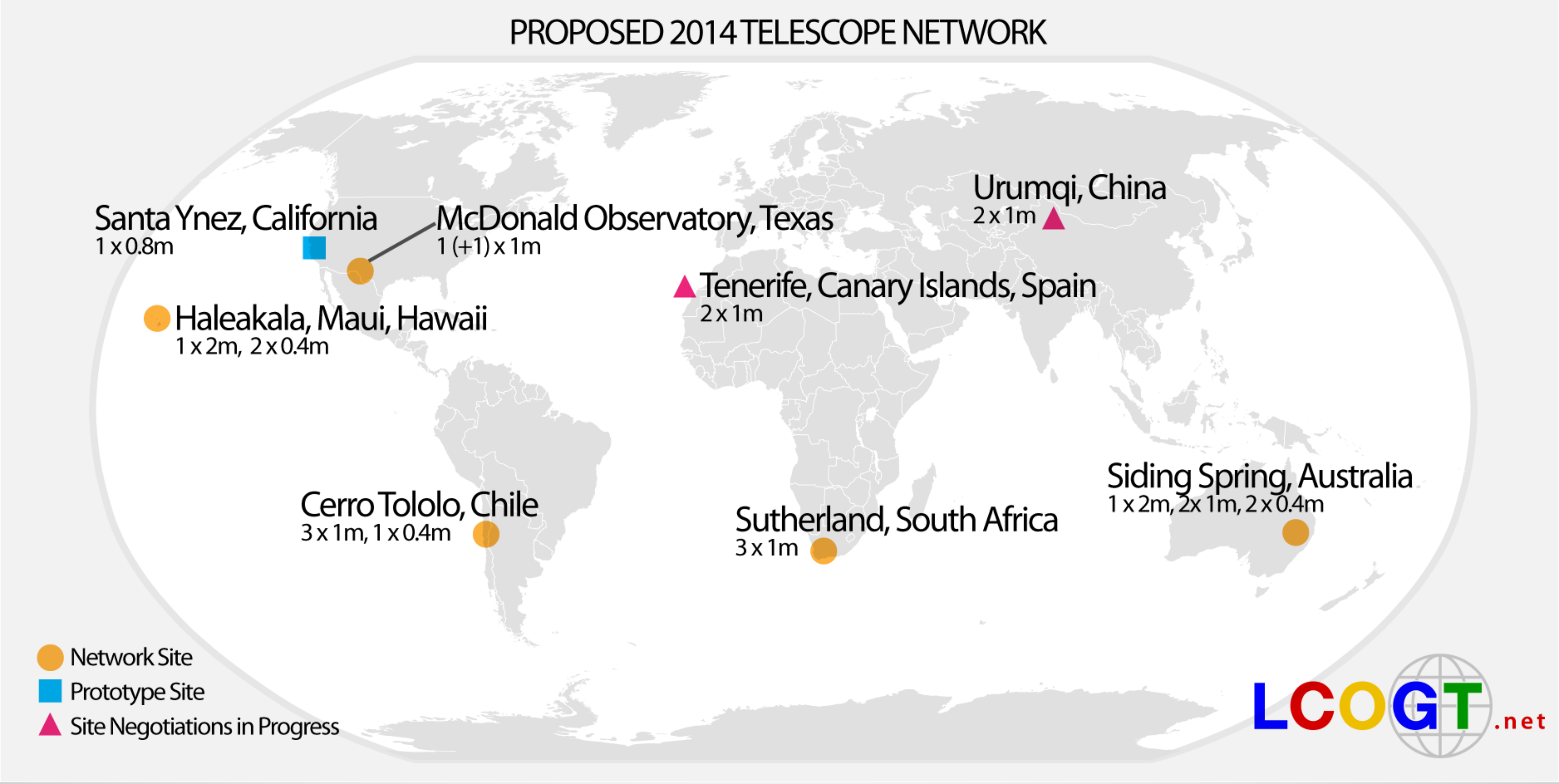
LCOGT.net

What is LCOGT?

- A non-profit organisation
- A professional observatory
- A network of longitudinally-spaced optical telescopes
- An ‘obvious’ idea
- A unique instrument for the time domain
- A new frontier in telescope scheduling



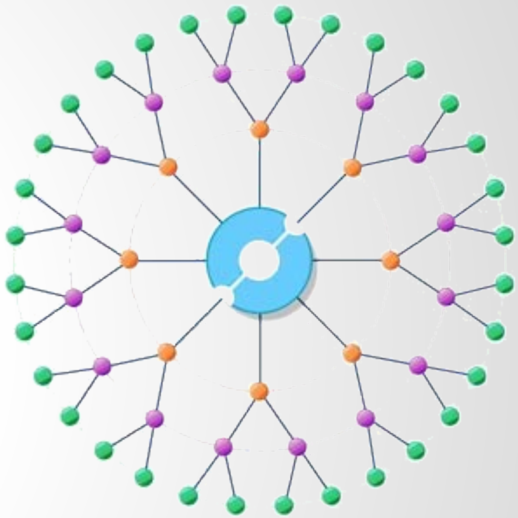
Keeping you in the dark



Cerro Telolo: 3 x 1m

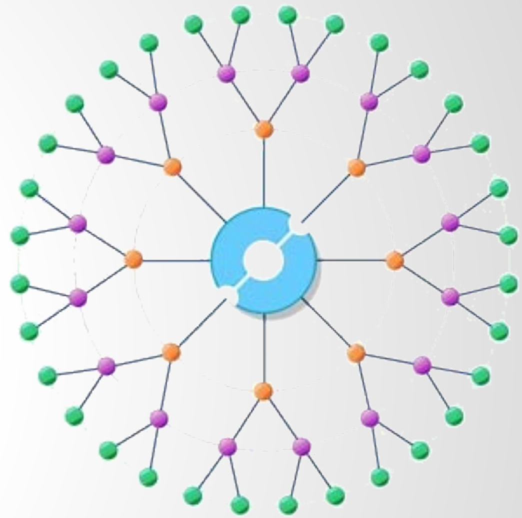
Network design philosophy (1)

- A single global observing abstraction
- Identical instrumentation
- Redundant resources
  - Cost-effective
  - Mitigates technical failures at site
  - Allows concurrent observing



Network design philosophy (2)

- Spectra and imaging at every site
- Fully robotic operation
- Tolerant to network outages
- Layered software intelligence
- Globally scheduled



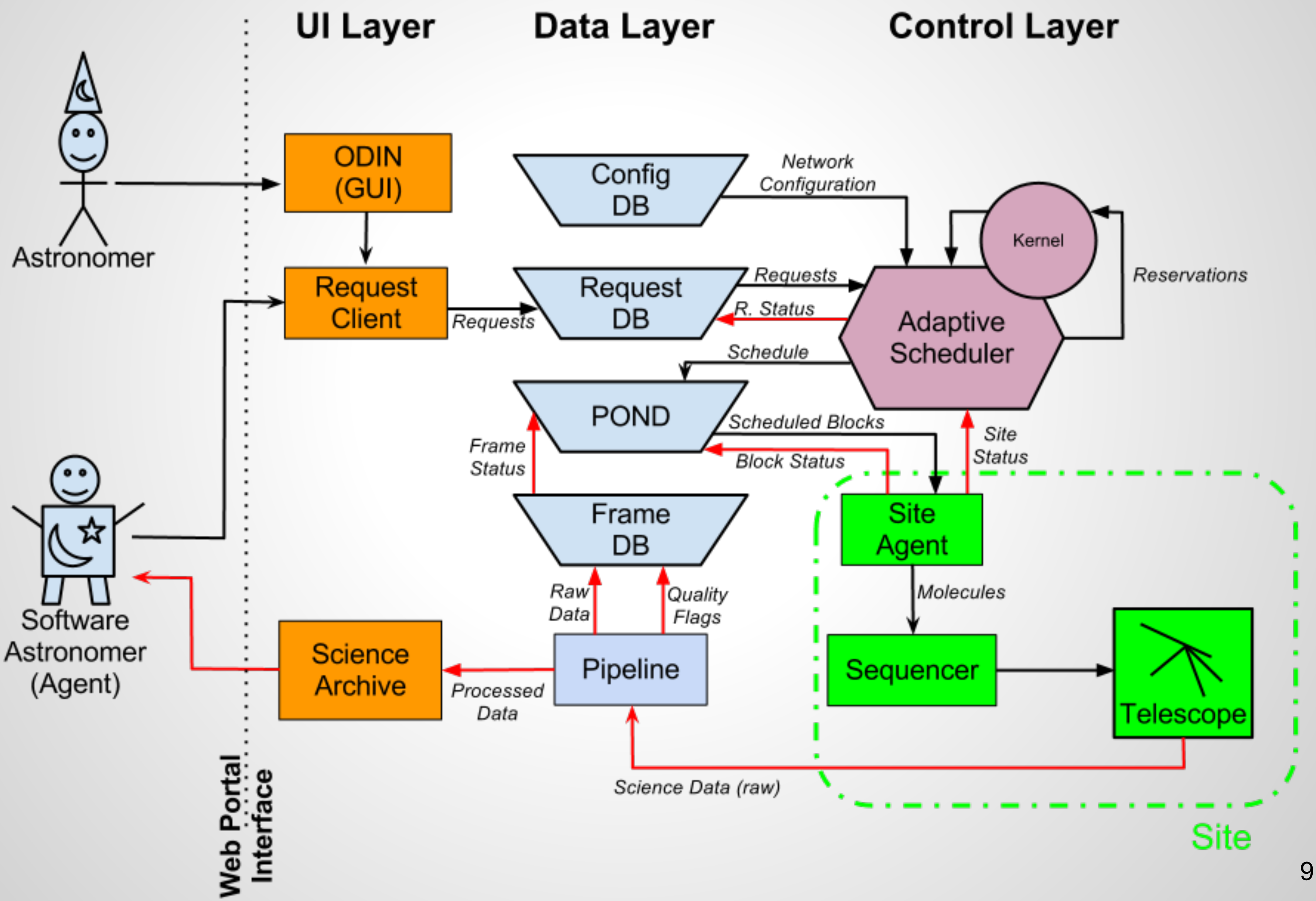
The (astronomy) problem we solve

- Accept requests from many users, at any time, from anywhere with an internet connection
- Support many different kinds of science
- Be highly responsive to new input
- Utilise the full potential of a telescope network
- Be a killer follow-up observing engine

LCOGT v1.0 capabilities

- Globally optimised observation placement
- Simultaneous and cross-site observing
- Automatic weather rescheduling
- Automatic hard constraint enforcement
- Automatic rescheduling of unsuccessful observations
- Target of opportunity observations, placed on-sky within 15 minutes of submission
- Human and machine request interfaces
- Cadence-driven, multi-site sequences
- Support for solar-system objects
- Low-dispersion spectrographs at both 2m sites
- Pseudo-real-time interface for education users

Architecture Overview



Robotic observing - a hierarchy

1. Scripted schedule (simplest)
2. Pre-computed schedule with human tweaks
3. “Locally optimal” dispatch scheduling
4. Lookahead solve (usually nightly)
5. Multi-telescope, multi-night global solve, with fast recomputes (*adaptive network scheduling*)

Can you formalise?

- Astronomical scheduling - it's not very good
- Telescope scheduling has no direct analog in classical CS or OR, but they are way ahead of us
- Formalise your problem, and you can leverage existing work
- Clearly distinguish between astronomy, abstraction, and implementation

What are your goals?

- Be very clear about goals
- Science? Efficiency? Airmass? Good seeing? Telescope wear? Completing cadences? Being fair?
- Not everything cool is feasible
- Desirable goals often conflict, sometimes irreconcilably
- *Scheduling* **(1)**: the art of deciding who loses

Make simulations!

- Simulation framework is time well-spent
- Keep it modular (you will need to revise it)
- Don’t over-simulate things that don’t matter (think spherical cows)
- Simulation is only as good as its input and assumptions
- Evaluation of schedule quality (simulated or real) is hard

Managing uncertainty

- *Scheduling* **(2)**: the art of deciding amidst uncertainty
- Dealing with uncertainty is hard
- Dealing with correlated uncertainty is very hard
  - e.g. statistical variations in weather
  - e.g. weird user submission behaviour
- Evaluating behaviour post-run with perfect hindsight is tricky

Many flavours of constraints and dependencies

- *a-priori* constraint*:* e.g. visibility, airmass, moon phase
- *real-time* constraint: e.g. seeing
- *scheduling-time* dependency
  - one request depends on another
  - one time depends on another (cadences)
- *post-completion* dependency
  - subsequent scheduling depends on post-completion function (*reactive scheduling*)

Common beliefs which are wrong

- local optimisation implies global optimisation!
- solving very large NP scheduling problems is impossible (airlines do it by magic)
- being fast enough to recompute an entire schedule in real-time is impossible
- science “requirements” (e.g. cadence goals, seeing) are clear, hard lines
- if you could just make your function a tiny bit cleverer, it will make people happy

Solving hard problems fast

- Large discrete optimization problems are usually solved in standard ways
    - linear/integer/mixed integer programming
    - constraint programming
    - a hybrid approach (e.g. branch and bound + constraints)
- Solving != **exhaustively prove**
- Solving == **good enough**
- Take a free grad course: Discrete Optimization (Coursera/University of Melbourne)

The evil of multi-objective functions

- heuristic by nature
- therefore messy, arbitrary and unsatisfying
- trade off one thing for another
- tend to “smear” outcomes
- simple objectives are rarely satisfactory
- complex functions tend towards making everyone equally unhappy
- if you have to do this
  - keep it as simple as you can
  - exhaustive simulation is essential

The human element

- justifying complex robotic scheduling to users/stakeholders is hard
  - top question: “why didn’t observation X happen”?
- people will attempt to twiddle even the most carefully nuanced scheduler, by hand
- humans need to build a mental model of your process before they will trust you
- stability in the continuum of schedules is highly desirable

The many flavours of cadence

"Observe 75% of this variable star's period" (exact, abstract by phase)
"Equally space these 10 observations somewhere good" (exact, abstract by space)
"Complete all 10 observations of this SN, or give up" (exact, all)
"Make a best effort to obtain 10 obs, but less is better than nothing" (exact, most)
"Make at least 5/10 observations, or give up" (exact, *n* of N)
"Make 4 observations, a day apart, somewhere good" (approx, abstract)
"Make 4 observations, a day apart, each +/- 6 hours" (approx, jitter)
"Make 10 observations, each between 3 and 6 hours apart" (approx, min/max)
"Complete all 10 observations of this SN, or give up" (approx, all)
"Make a best effort to obtain 10 obs, but less is better than nothing" (approx, most)
"Make at least 5/10 observations, or give up" (approx, *n* of N)
"If an observation failed, observe again as soon as possible" (approx, continue on fail)
“Observe this target 20 times, with logarithmic spacing” (non-linear cadence)

Random grab-bag

- retrying on failures has multiple subtleties
- state gets complicated quickly
  - hard to understand
  - expensive to maintain and fix
- automated tests (unit, integration) are essential if you want to improve safely
- build modularity everywhere: you won't get this right
- ToOs are a pain, and need special consideration

Main takeaways

- It is possible to solve very large scheduling problems quickly, with the right formalism
- What do you want?
- Do simulations!
- It is not sensible to optimise **everything**
- Uncertainty, multi-objective functions, fast recomputes, cadences etc. are **hard problems**
- How people perceive your stuff is important
- The output of a clever scheduler will be beyond your understanding - good metrics are crucial

Questions?

Identifying the formal problem

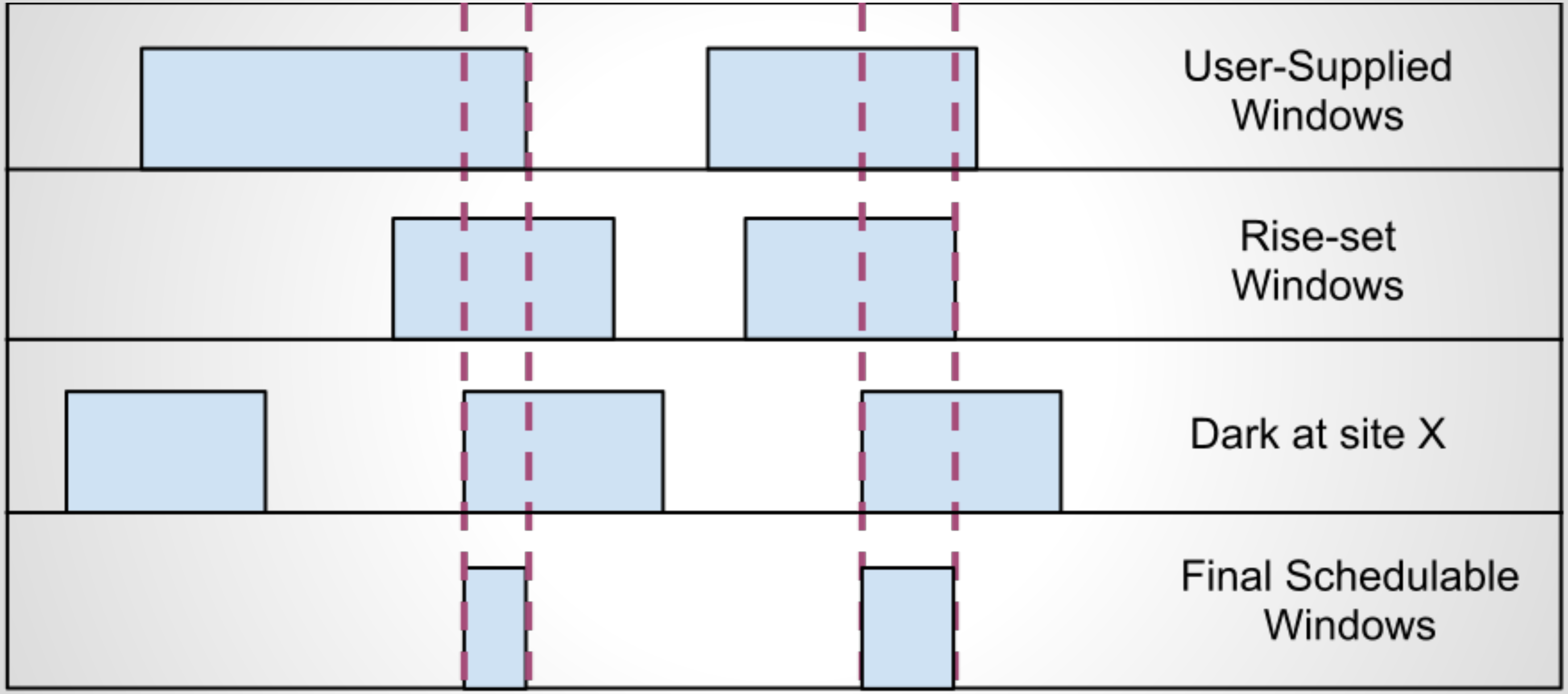
- Decomposed the problem
- Expressed the problem formally
- Checked it hadn’t been solved
- Published the formalism (the *telescope network scheduling problem*), for others to attack
- To find a practical solution, devised various approximate transformations into problems that can be solved with known techniques

Common features of the formal problem

Four key phrases for literature searches:

- **Interval scheduling** (non-overlapping discrete time windows)
- **Slack** (flexible start/end times)
- **Multi-resource problem** (concurrent, not interchangeable)
- **Scheduling-time dependencies** (logical connectors)

Astronomy by Intersection



Moon distance or other *a priori* constraints would be applied in the same way