The “Terrapin” algorithm

Fully optimal scheduling, obeying constraints

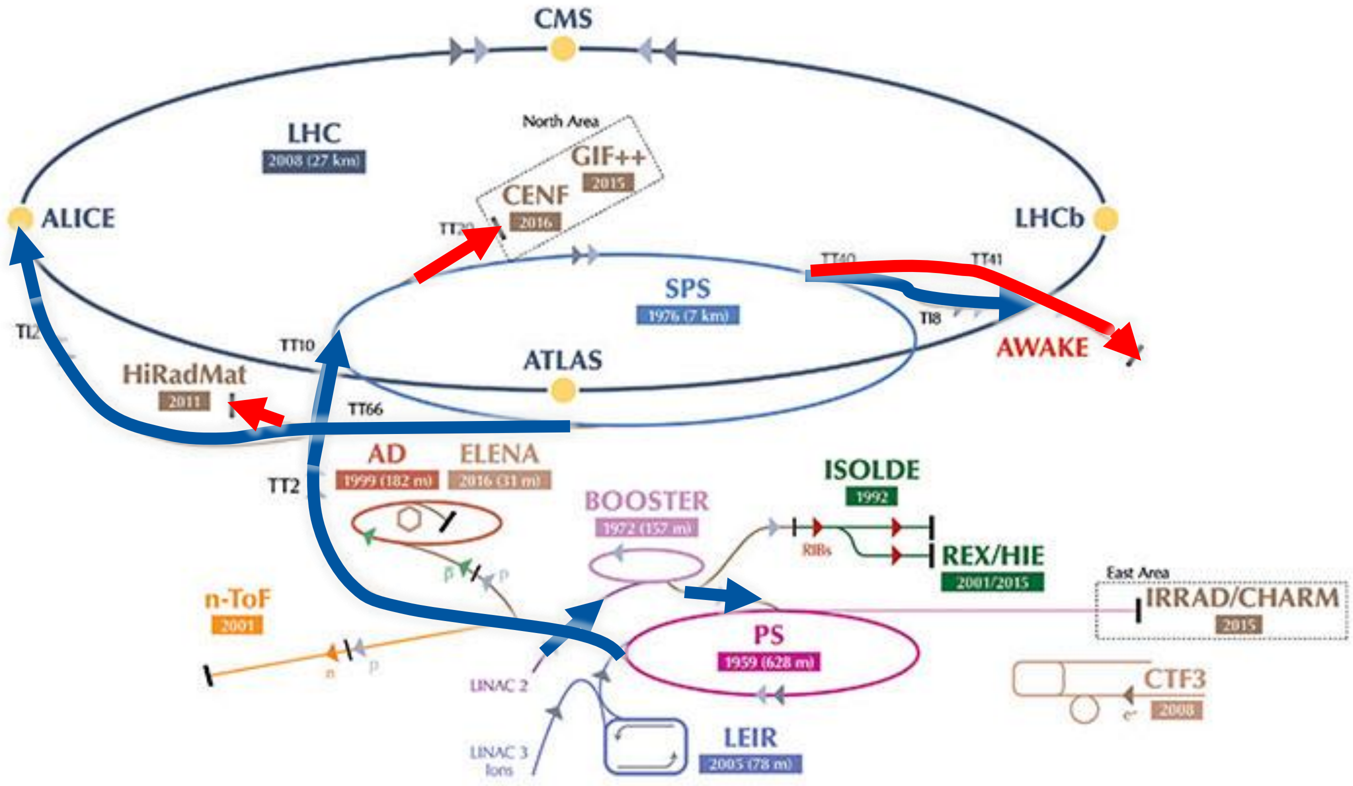
Seminar 1 (of 2): Non-parallel scheduling

Sandy Easton (BE-OP-PS)

Chapter 1

Design context for the algorithm

Motivation



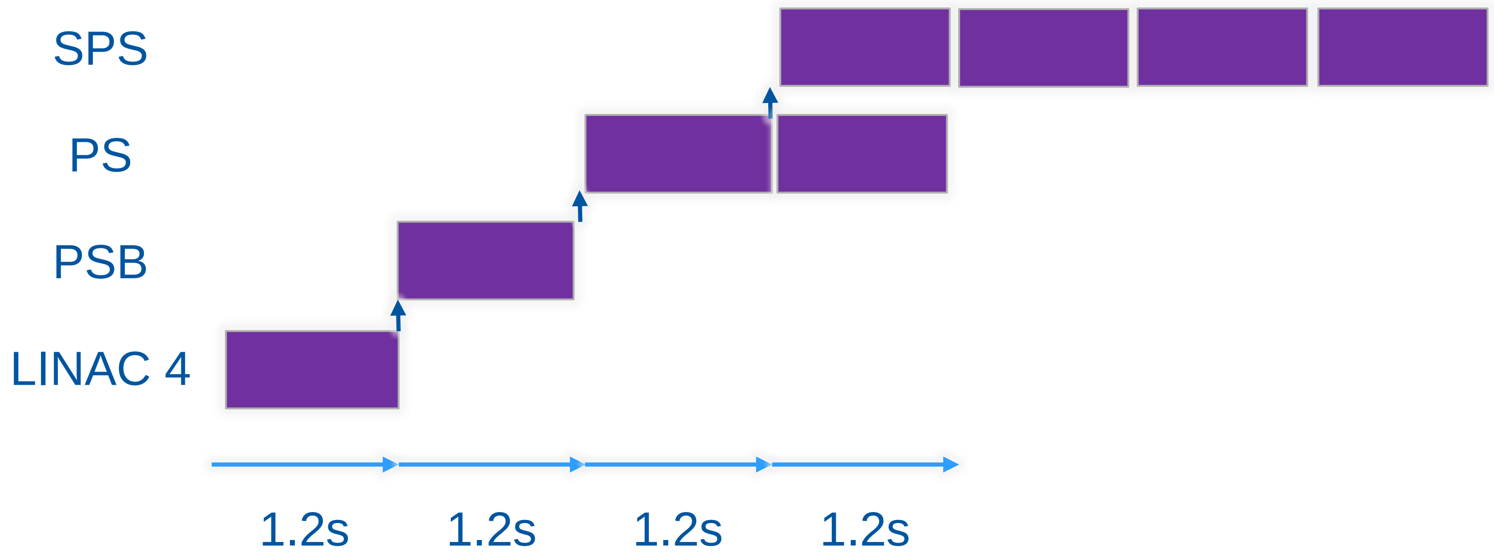
Key

Transfer line

Experiment line

The master timing system

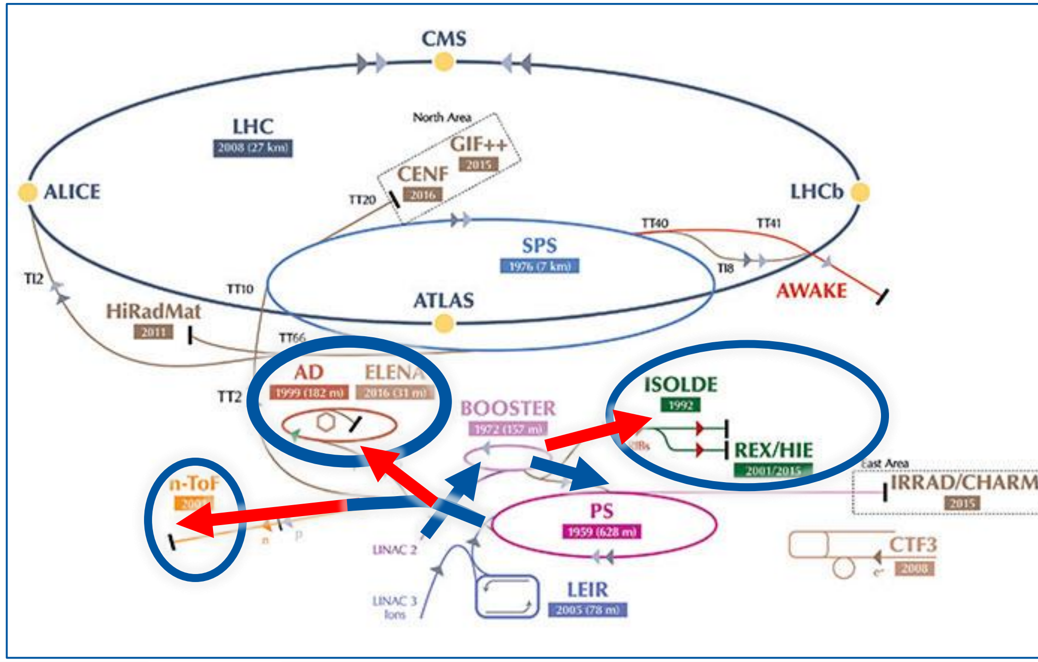
- CERN's master timing system uses quantised time periods of 1.2 seconds
- This is to provide a common "beat" for synchronising accelerators
- Any beam will occupy a whole number of these "basic periods" in each accelerator.



Beams are made of "Tetris" style 2-D blocks

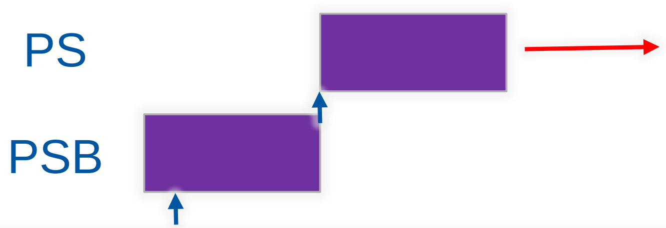
Lower-energy users

- Many experiments do not need the entire injector chain

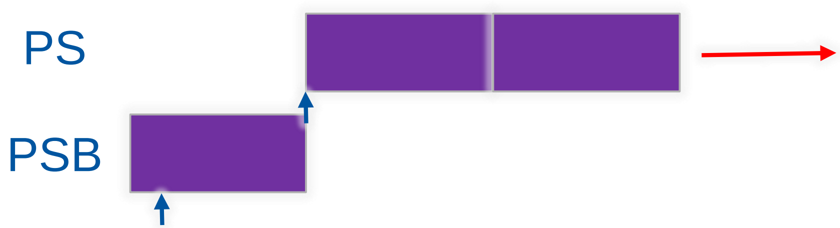


Examples:

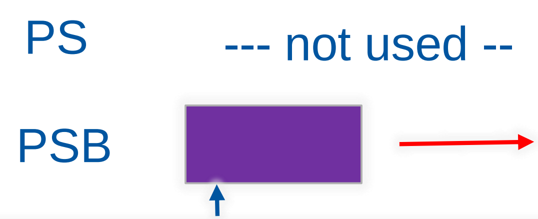
nTof



AD



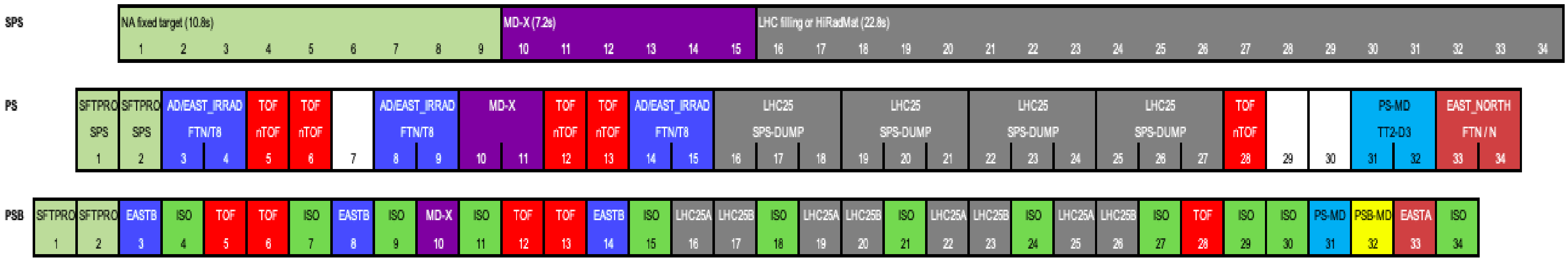
ISOLDE



All users can only be furnished between injections to larger accelerators

The Supercycle

- The “Supercycle” is the beam schedule for the whole injector chain



- It is “cyclic”:
  - Once finished, it immediately repeats
  - It has a finite length (defined the highest-priority users)

- You may be able to recognise some beams structures in there….!
  - But if not… it’s not important for todays purposes.

Context Summary

The key points for understanding today's algorithm explanation are

1. The term "supercycle" is just the special name for the schedule we use
2. The supercycle is made of quantised time periods ("Basic periods")
3. The supercycle has a finite length
4. The supercycle repeats itself once it has completed

Constraints

- There are many constraints on how beams can be follow each other.
- A supercycle which doesn’t respect any constraints is simply not valid.
- Involved departments/users/operators have been polled extensively to collate the constraints required to satisfy our operations

Constraints

Examples

Hardware constraints

- Magnet switching times
- Radiation limits of zones
- Power supply RMS currents

User constraints

- Stray-field avoidance between accelerators
- Always/never following certain beams (hysteresis avoidance)
- Experiment repetition rates (max, min or sometimes both!)
- Required number of instances of each beam (max or min)

This list of “real” constraints aren’t exhaustive...
… but the following one of “abstracted constraints is thought to be

Constraints

- All reported constraints can be categorised into certain cause/effect types

Cause types

- "Unary"
  - Placing an instance of a beam has an effect on how another can be placed
    - E.g. Beam X must not be within 3 BPs of Beam Y

- "Cumulative"
  - The total number of instances of a beam violates a constraint (or not)
    - E.g. RMS magnet currents averaged over the supercycle
    - E.g. Required number of instances of beam X

Constraints

- All reported constraints have the following characteristics:
  1. Either “Unary” or “Cumulative” in their cause.

<u>Effect types</u>

- <u>Boolean</u>
  - *The placement of beam X allows (or disallows) the placement of beam Y*
    - E.g: requesting to only immediately follow certain beams

- <u>Offsetting</u>
  - *The placement of a beam X delays when beam Y can next start*
    - E.g. Avoiding hysteresis left by earlier beams.
    - E.g. Waiting between subsequent shots to an experiment (Y=X)

- <u>*Max-Offsetting*</u>
  - *The placement of beam X restricts when beam Y must start before*
    - E.g. Constant-flux fixed target experiment: No more than 10s between shots

Multiple Constraints on beams

- Operations are not limited to having single constraints from/on individual beams
    - Numerous constraints can apply to a single beam
    - A single beam to cause constraints on multiple other beams

- This is no problem

Multiple Constraints on beams

Examples

- Offsetting & max offsetting, all at once
  - "Beam Y must be between 3 & 7 BPs after Beam X"

- Boolean/Offsetting/Max-offsetting from multiple beams
  - "Beam Z must be at least 4 BPs after Beam X and at least 7 BPs after beam Y"
  - "Beam Z can only follow Beam X or Beam Y"

- Boolean constraints combined with offsetting/max-offsetting
  - "Beam Z can only come immediately after beam Y & be >4 BPs after Beam X"
    - This is fine but causes a few extra complications (not discussed today)

Inputting constraints into the algorithm

- The algorithm **does** need to **obey** constraints
- It **doesn't** need to **understand** the reasons behind them
    - Indeed, an algorithm that could would virtually be an AI accelerator operator….

- A UI could be built to understand magnets, beamlines destinations etc.
    - But this is a huge task itself …
    - … on top of
        - designing this algorithm
        - coding this algorithm
        - making a rudimentary UI for demonstrations and testing

- The existing UI relies on operators to interpret algorithms from "real" constraints to their "abstracted" functions.
    - Operators already chiefly deal with abstracted constraint functions anyway.

Inputting constraints into the algorithm

<u>Example of a real constraint</u>

- A magnet called BHZ.377 takes about 1.5s to change from steering beams down a beamline to steering them down the "FTN" beamline (and vice versa)
- Suppose beam X goes down the FTN line, & beam Y goes down the TT10 line.

<u>What does the algorithm need to know?</u>

- If we place X, it is illegal to place Y without having an offset of 1 BP
- If we place Y, it is illegal to place X without having an offset of 1 BP

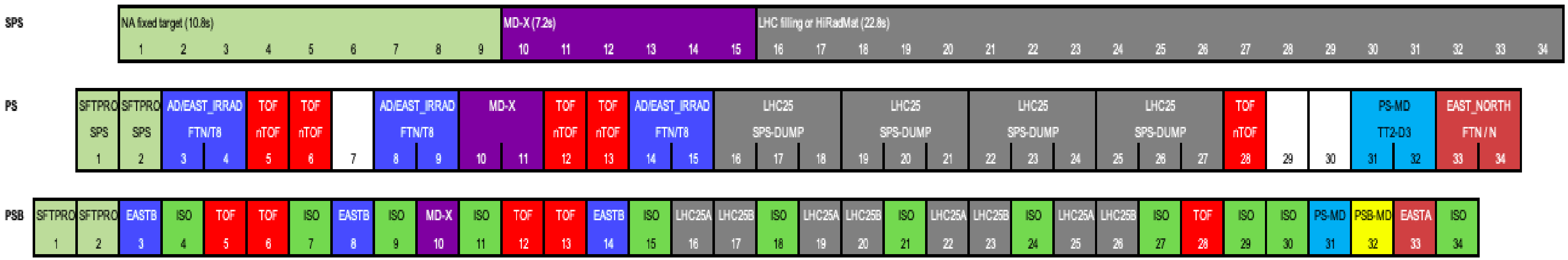
↑

Abstract constraint

- Every "real" constraints translates to an "abstract" constraint
- The algorithm uses these abstracted constraints
- How the abstraction is done is a wider question for our whole team

Final word on constraints

- Recall: The supercycle is "cyclic"



- A unary constraint cause by a beam at the end of the supercycle...
  …. affects beams at the start of the supercycle

- However, the constructive search process is done from "start" to "finish"
- It is not possible to know the "looped back" constraints on beams when starting the search
- Ensuring "looped back" constraints are not violated requires a special extra step
  - It is called the "*loopback check*"

Summary of key points

- The key points for understanding today's algorithm explanation are

1. The term "supercycle" is just the special name for the schedule we use
2. The supercycle is made of quantised time periods ("Basic periods")
3. The supercycle has a finite length
4. The supercycle repeats itself once it has completed
   - Requires the constraints "loopback" check
5. The algorithm uses the "abstracted" function that underlies any "real" constraint
6. Categories of abstract constraints:
   - Causes:
     - Unary
     - Cumulative
   - Effects:
     - Boolean
     - Offsetting
     - Max offsetting

Chapter 2

Scope of today's seminar

Scope of today's seminar

1. Today, we **will** explore how the algorithm
   - comprehensively explores the schedule space for a *single*, *uncoupled* accelerator
     - i.e. "is as strong as a brute-force"
     - this is quite a unique feat…
   - ensure all constraints are obeyed
   - outputs a truly "optimal" schedule
2. Today, we **will not** explore how the algorithm
   - Performs this search for accelerators in parallel
   - Deals with "Spares"
     - Not as complex as it may appear
   - More detail on some constraints
   - Works even faster than shown
     - The "2.0" is orders of magnitude faster than demonstrated today
     - … but 1.0 achieves exactly the same results

will be the subject of a second seminar

What is special about Terrapin?

- Many good scheduling algorithms exist already
  - … many can work with convoluted sets of constraints
  - … but they are invariably heuristic algorithms
    - Genetic, heuristic repair, particle-swarm, etc.…
    - Machine learning

Why are they “invariably” heuristics?

- Because the set of all schedules (even for small scheduling problems) is HUGE
  - It is a permutation space
  - WRONG: it is even bigger than a permutation space….!

- Heuristics use educated guesswork to avoid looking at all the schedules
  - Keeps runtime practical
  - Speed at the expense of thoroughness

- Good heuristics are indeed powerful, useful algorithms, but….
  - you can never be sure if their solutions are indeed optimal
  - If they cannot find a solution, it doesn’t necessarily mean that no solution exists

What is special about Terrapin?

Answer

1. It will find the truly optimal solution
   - It does not perform a brute-force…
   - …but a stronger solution than it finds does not (and cannot) exist

2. If it finds no solution, then it is certain that no solution exists

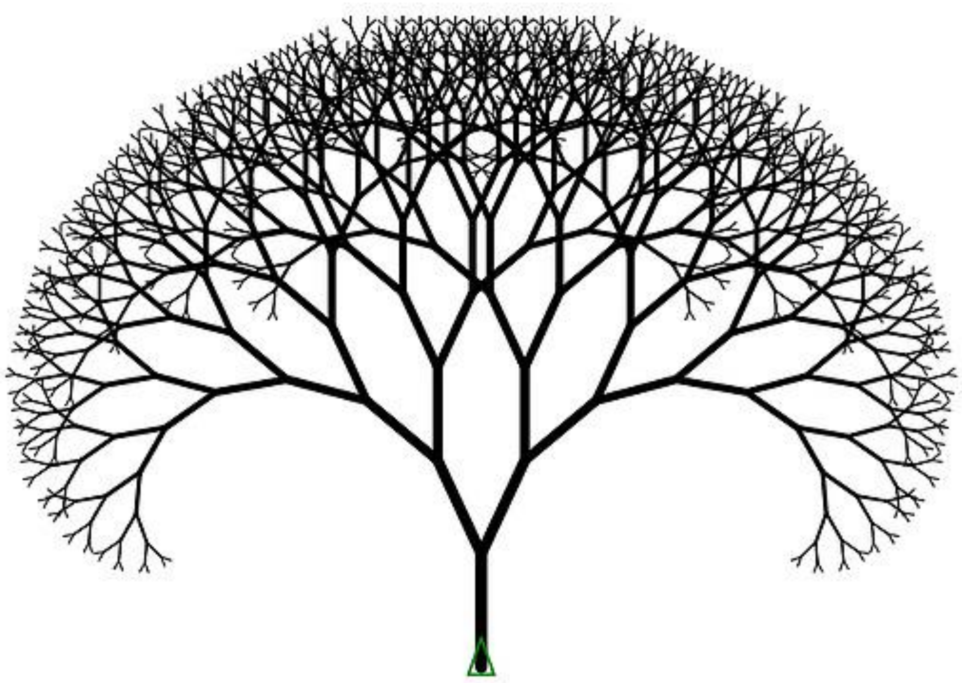
- Terrapin keeps a practical runtime, while not suffering the weaknesses of rigour which heuristics heuristics have

Chapter 3

Foundational ideas

Main idea

- A brute-force becomes so unmanageable because of the exponential growth seen within the search space



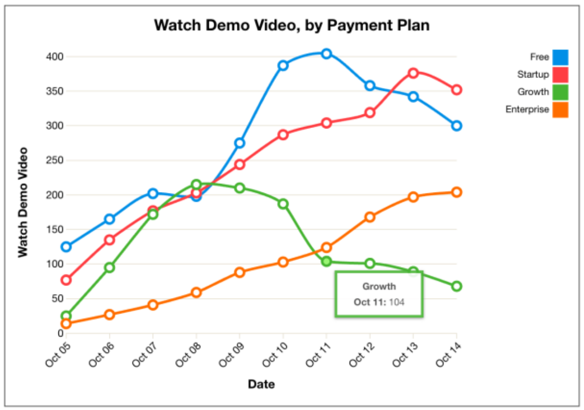
- This tree shows the number of 10-long permutations of two objects
- The possible permutations to be compared are the 2^{10} = 1024 “leaf nodes”
  - For schedules, not permutations, there would be even more than 1024 nodes

Main idea

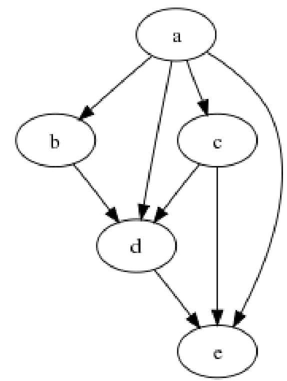
- Terrapin would perform the same schedule-search by creating a much neater graph
    - The graph would contain 21 nodes, and would contain 1024 *paths* through it

N.B. The word “graph” has two meanings:

1: A chart



2: A network



← This type

Analogy

- There is not a route between your work and your home that you wouldn’t understand…
- … but in your whole life you couldn’t drive (or list) all of them….!

Trimming techniques

Topological simplicity

- It was mentioned earlier that a "schedule space" is larger than the space of permutations of the items it schedules.

- Even with constraints, at any point in building a supercycle there is always an *earliest* next place that any requested beam can go

- Nothing can be gained by placing anything *later* than this earliest place
  - This would give a different schedule, yes….
  - … but not one you need to look at when looking for an optimal solution!

- Terrapin only considers the "topologically simple" placements.
  - It never adds any space that is not absolutely necessary

You can draw a supercycle that Terrapin would never find…
… but it would be equivalent to a "topologically simple" one which Terrapin did find

Completeness without exhaustivity

Trimming techniques

Constrained Construction

- Most constrained-scheduling algorithms obey constraints by
  - Building a schedule (or sub-schedule)
  - Checking if it obeys all the constraints
  - Discarding it/trying to repair it if not.

- Terrapin constructs it's graph one placement at a time
  - It doesn't ever look at 'potential' sub-schedules of multiple placements
  - No placement decision is ever made which violates constraints from other placements (/nodes)

The paths within the Terrapin graph all have no violations of any constraints

- A path within the Terrapin graph represent a topologically simple schedule
- This schedule is guaranteed to not violate any constraints.

What is Optimal…?

"One person's rubbish is another person's treasure"

- A supercycle is *"valid"* if
  - it violates no constraints
  - it satisfies all the minimum requests

- A supercycle is "*optimal*" if
  - it is valid
  - no preferable supercycle is possible

- Which users should be allocated any available "extra beams" above the minimum depends on *"preference"* of current operations

- Terrapin computes the set of allocations possible from all valid supercycles
  - The "optimal" is the highest member of this set…
  - … according to your own particular priority ordering

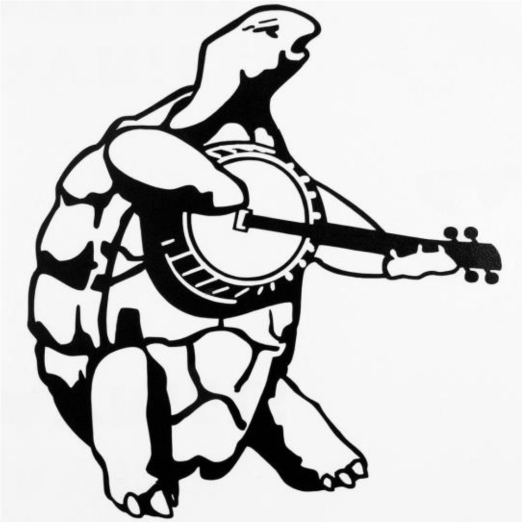
Key Points

1. The paths within the Terrapin graph contain no violations of any constraints
    - The are all completely valid solutions

2. The definition of the “optimal” solution depends on which users you prioritise

Chapter 4

The Terrapin Algorithm: Part 1



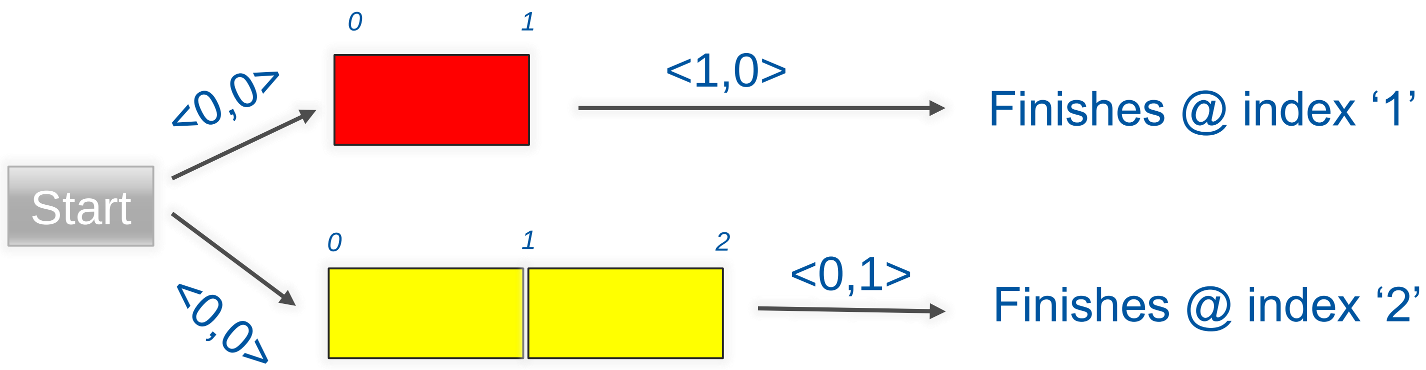
Example 1

- Two beams
- Supercycle is 4 basic periods long
- No constraints

Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

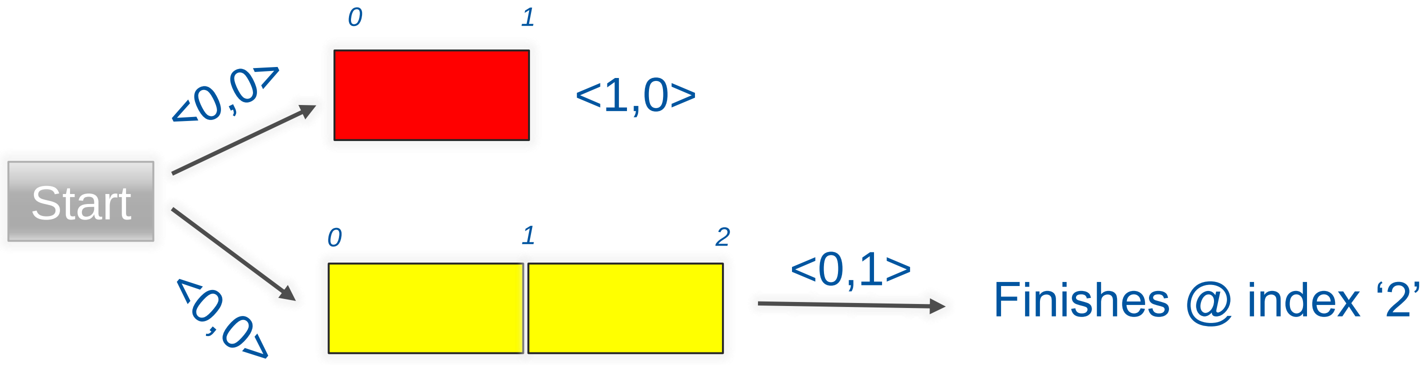
Phase 1: Building the graph



Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

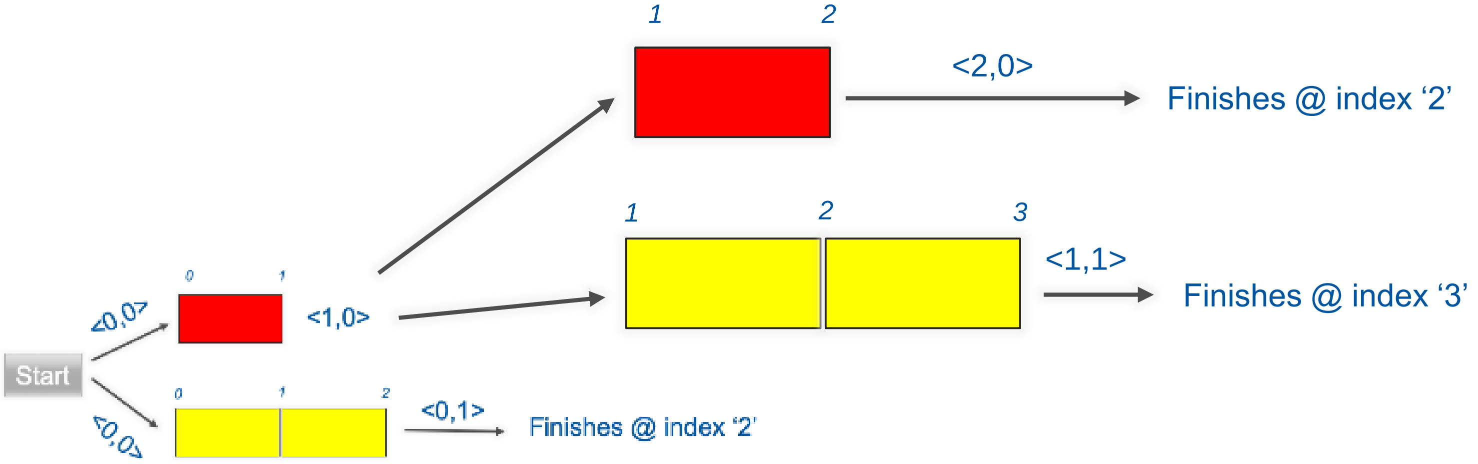
Phase 1: Building the graph



Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

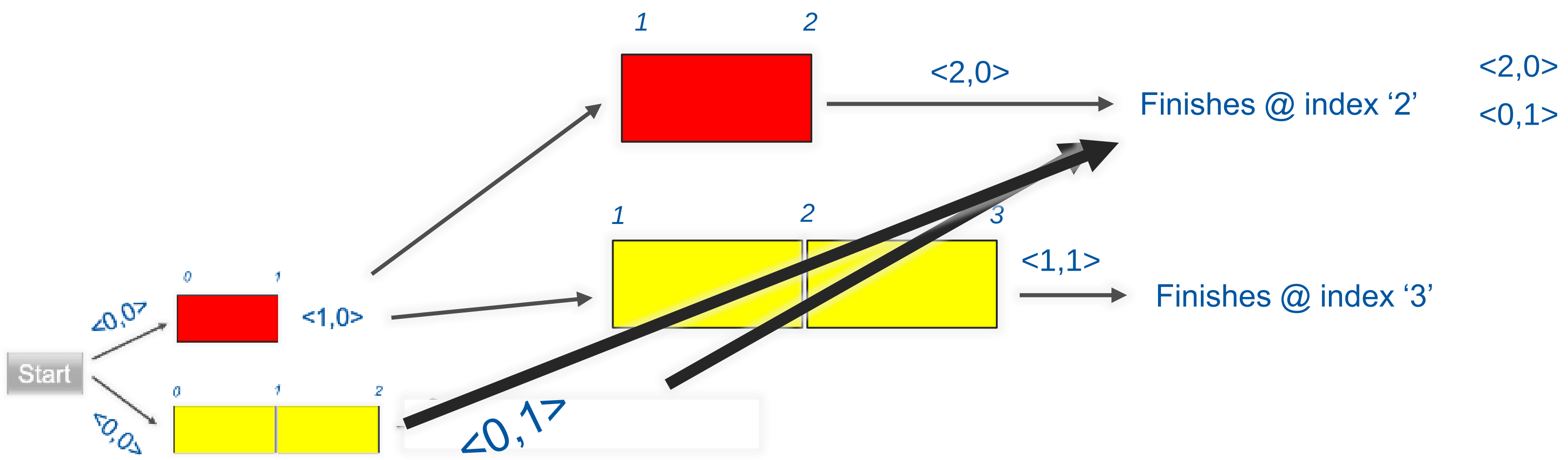
Phase 1: Building the graph



Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

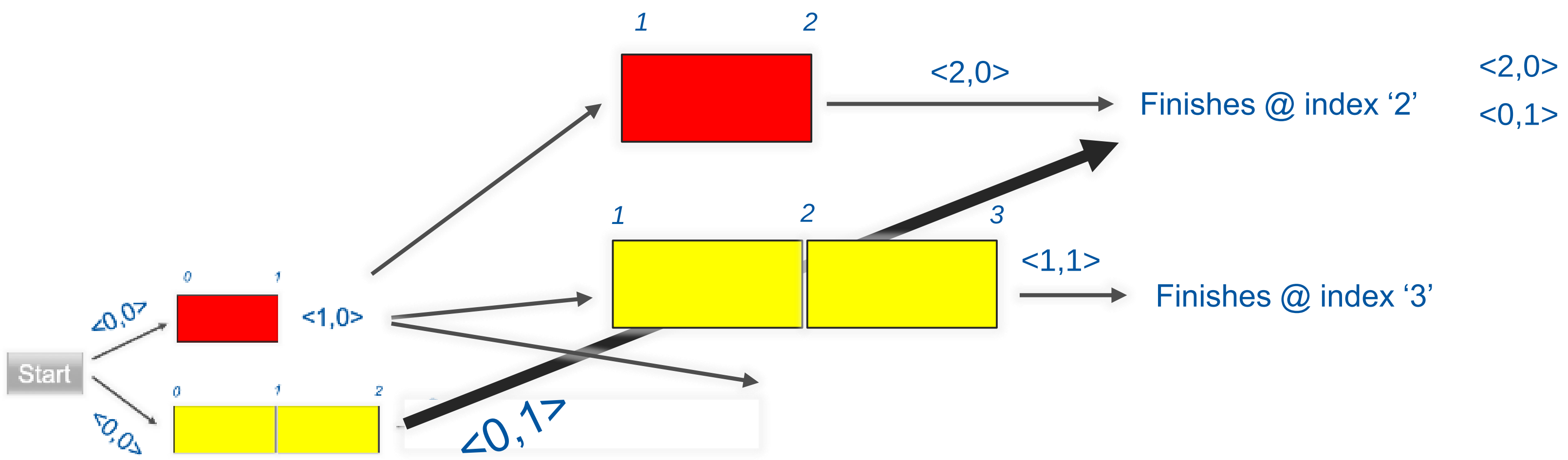
Phase 1: Building the graph



Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

Phase 1: Building the graph



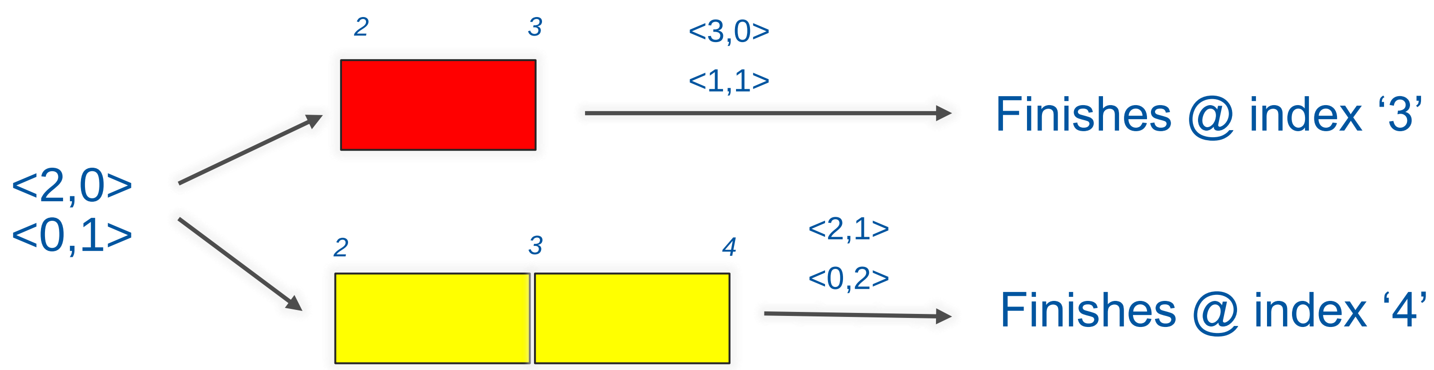
Next index is index 2…

Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

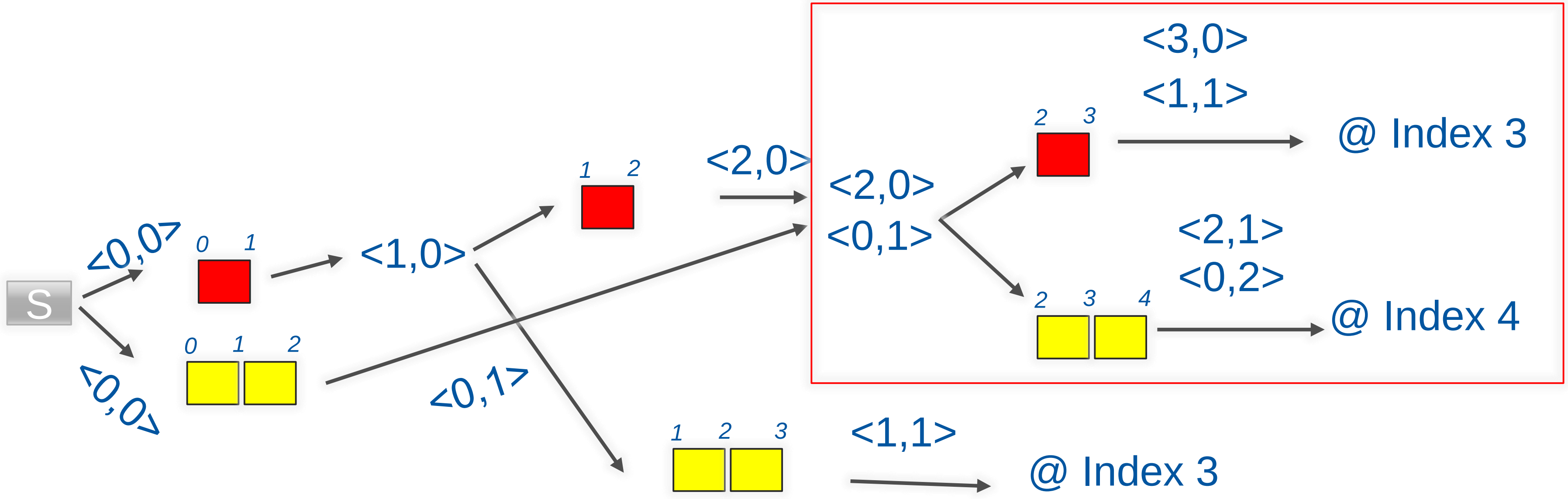
Phase 1: Building the graph

Building on the options discovered to give index 2



Example 1

- Two beams
- Supercycle length = 4BP
- No constraints



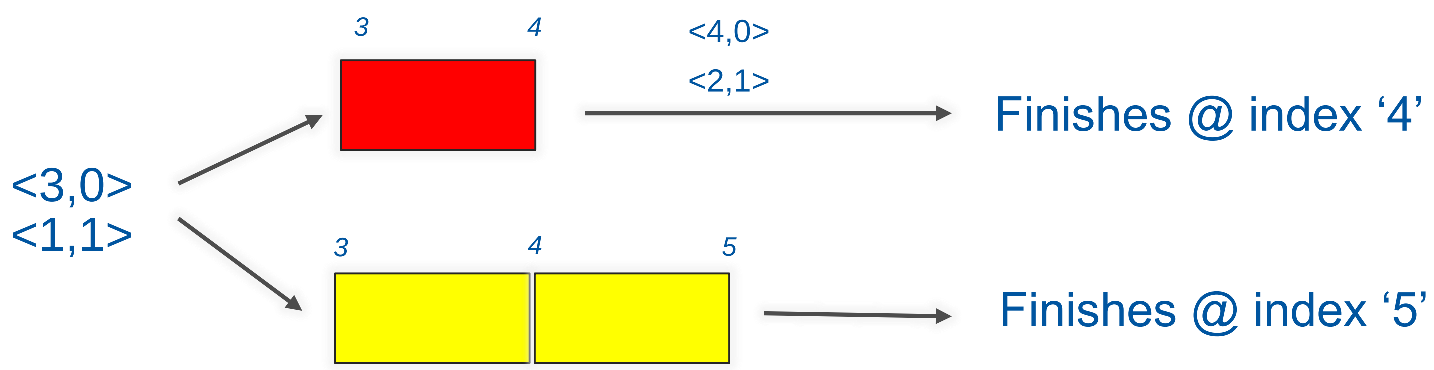
- Next is index 3….
- Notice <1,1> has appears twice for index 3...

Example 1

- Two beams
- Supercycle length = 4BP
- No constraints

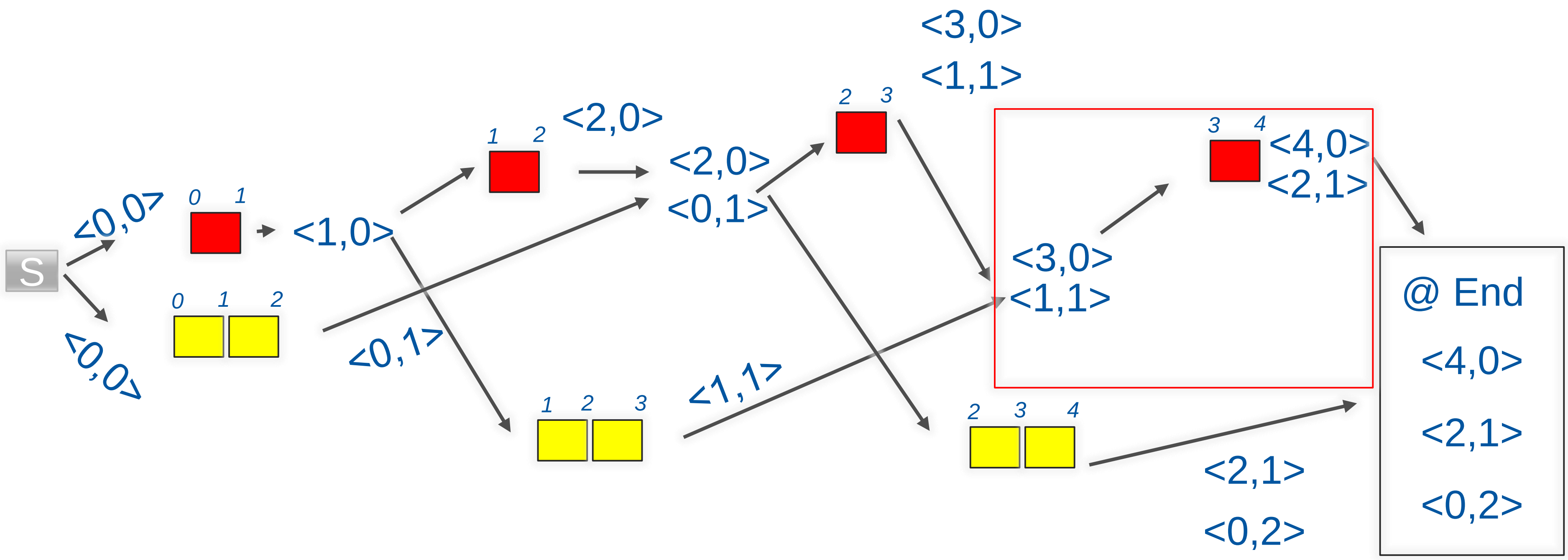
Phase 1: Building the graph

Building on the options discovered to give index 3

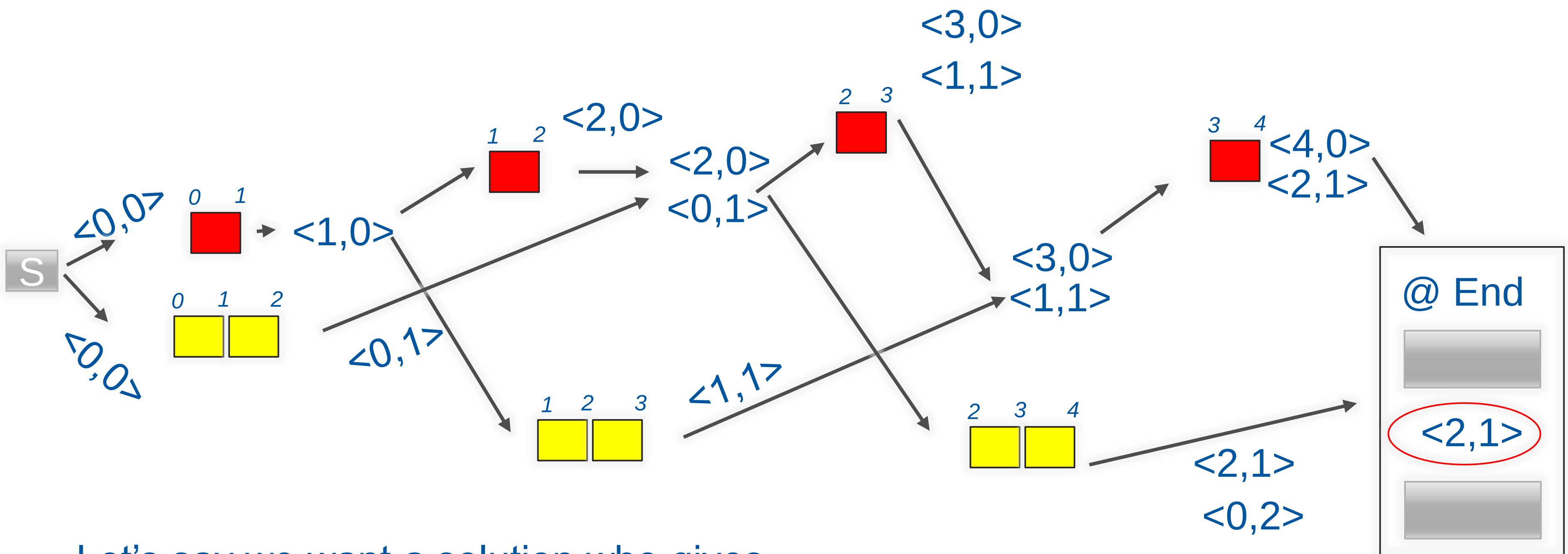


- This goes out of the bounds of the supercycle….
- …so we needn’t consider it
- Indeed, next building index is index 4 – the end of the supercycle
  - We have finished building

Example 1's completed graph



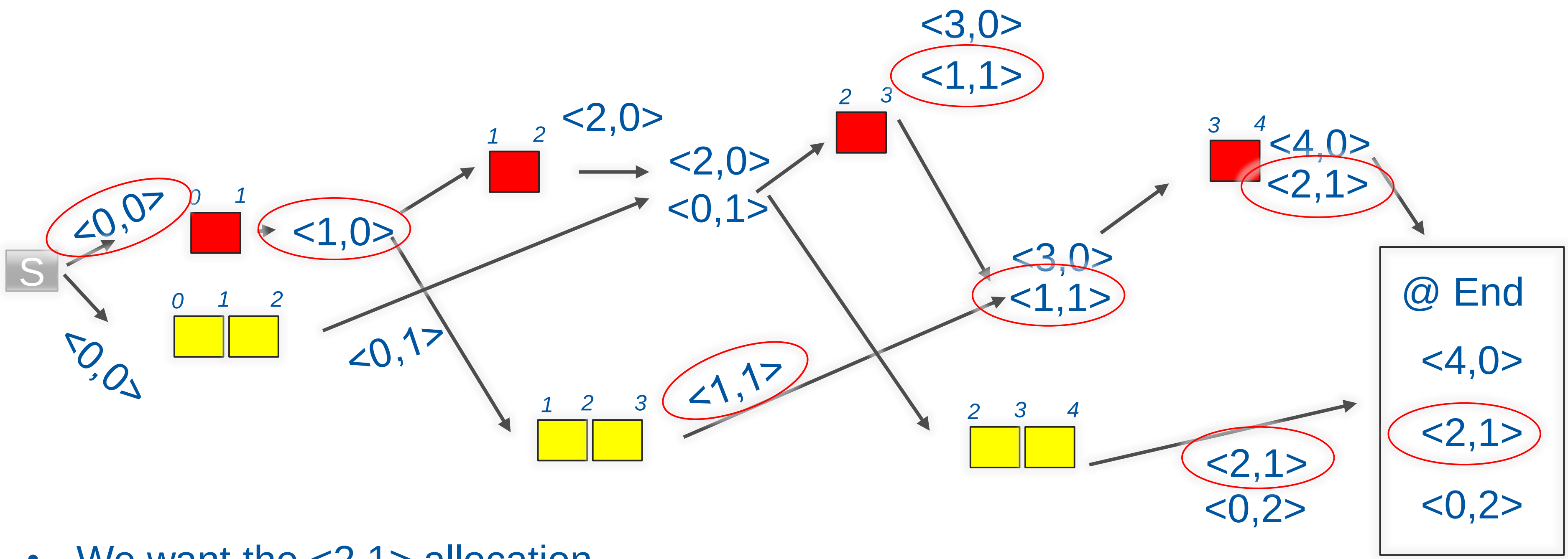
Example 1: Building a solution



- Let's say we want a solution who gives
  - at least 1 instance to each beam
  - Prioritises B over A for extra beams

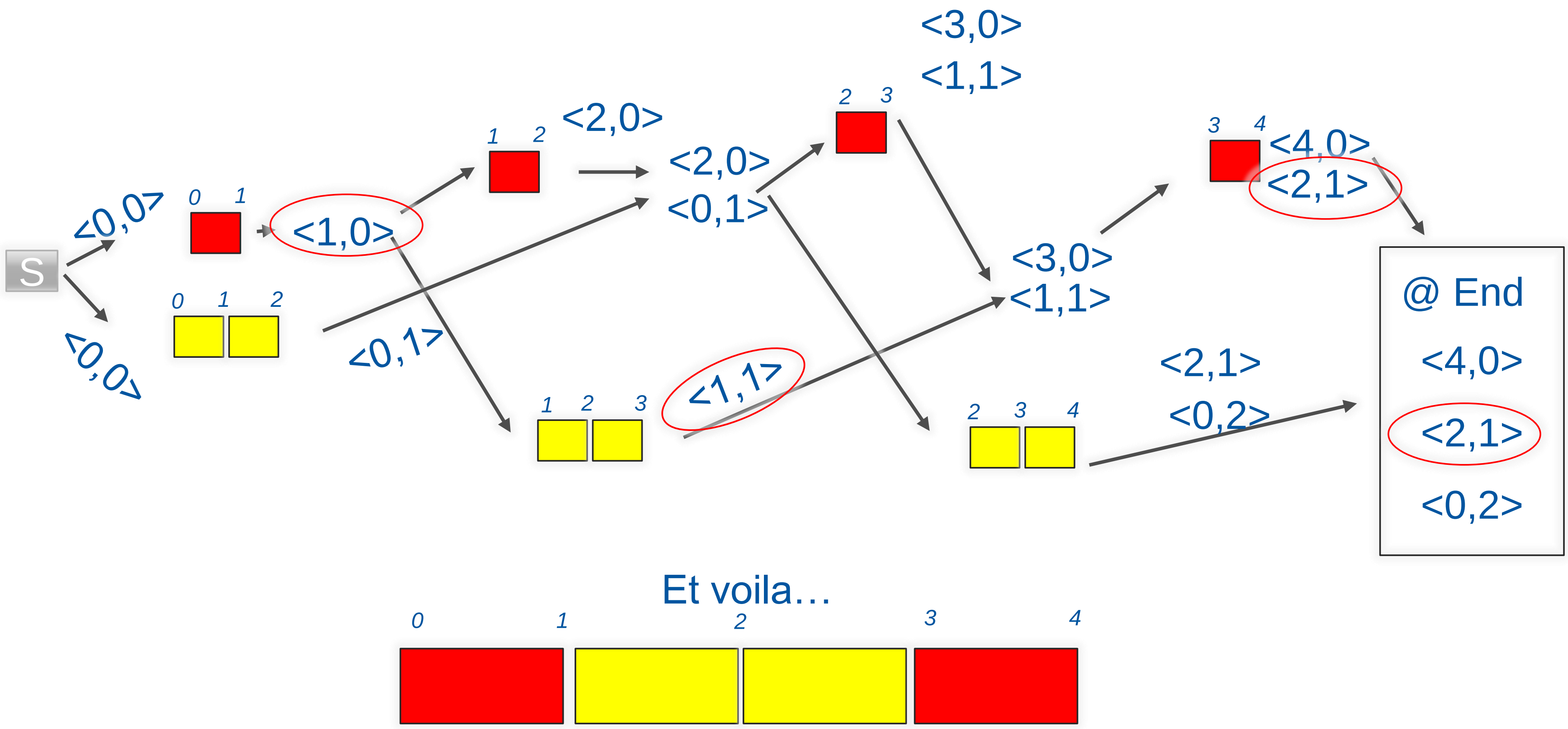
- We want a schedule with <2,1>
  - This is optimal for us

Example 1: Building a solution



- We want the <2,1> allocation
  - We have 2 options for <1,1>: we can pick either
    - Chose A @ 3: <2,1> should decrement to <1,1>
  - We have 2 options for <1,1>: we can pick either
    - Choose B @ 1: <1,1> would decrement to <1,0>
  - Only one seed for <1,0>: -> A @ 0
  - We're now @ 0: Done!

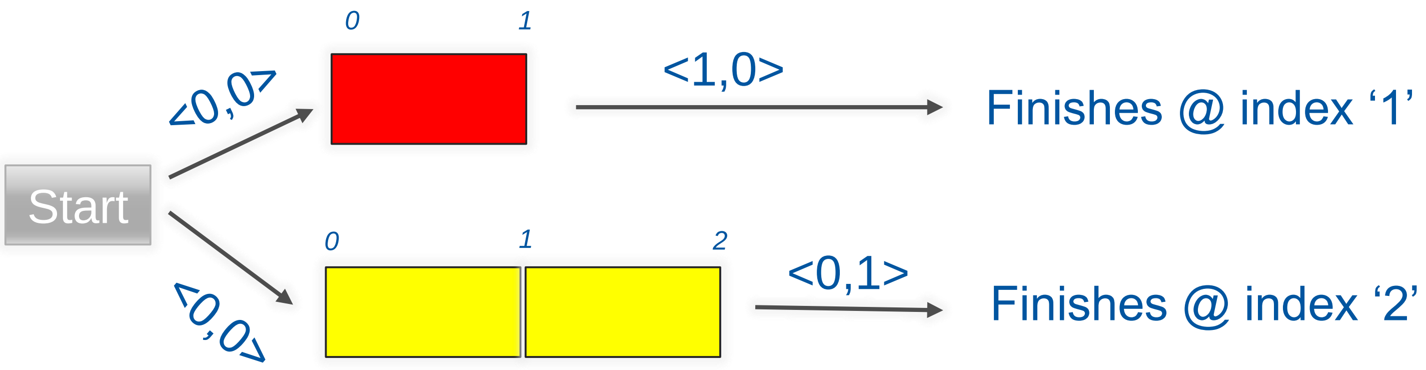
Example 1: Building a solution



- Any other supercycle is a path back through the graph like we just did
- The only allocation arrays possible are the ones you see on the right

Example 2

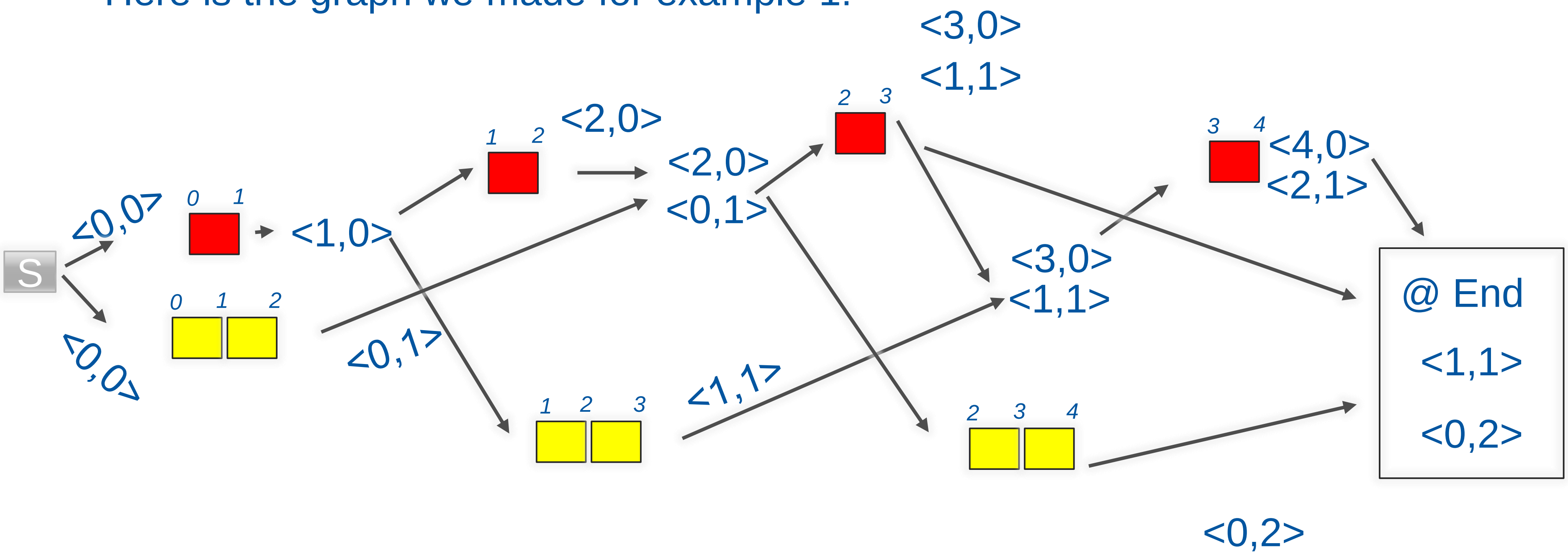
- Same setup, but let's say A can only come after B
- We started example 1 like this:



- However, we haven't placed anything at the start….
  - So the boolean constraint for allowing A is not triggered yet

Example 2's completed graph

- Here is the graph we made for example 1:



- Similar logic to what we just saw would cause the graph to be built differently
- Many connections would not be made
- And many allocation calculations would not happen
- We see a different set of possible allocations
    - But every path is still a valid supercycle

Time complexity

- Terrapin runs in quasi polynomial time
  - Not exponential time like a brute fore
- The maximum work done at each step does not increase as theschedule length increase
  - Unlike a brute force

Further applications

Non-scheduling problems

- Can be readily applied to other scheduling problems, of course…
- …but it is foreseen that the process can be abstracted to work on many other sequential processes where the current state does not depend on the *order* of previous states
  - Many card games
  - Scanning the parameter space within sequential finite-element models
  - Chess….!?! Hmmm, still no….
- Exciting!
  - Expecting a second collaborative publication formalising types of further applications

As part of a hybrid optimiser

- For huge problems, Terrapin can be thought of as a sub-optimiser
  - Potential for creating a very powerful hybrid

Training-example generator for learning algorithms

- Terrapin's output is an orderable list of objective function values
- Each objective function value has a large list of concrete solutions behind it

Questions

And see you for part 2!