Streaming Auto-Scaling in Google Cloud Dataflow

Manuel Fahndrich

Software Engineer
Google

https://commons.wikimedia.org/wiki/File:Globe_centered_in_the_Atlantic_Ocean_(green_and_grey_globe_scheme).svg

Individual Ranking

Sarah	151,365
Joe	109,903
Milo	98,736

⋮

Team Ranking

	1,251,965
	1,019,341
	989,673

⋮

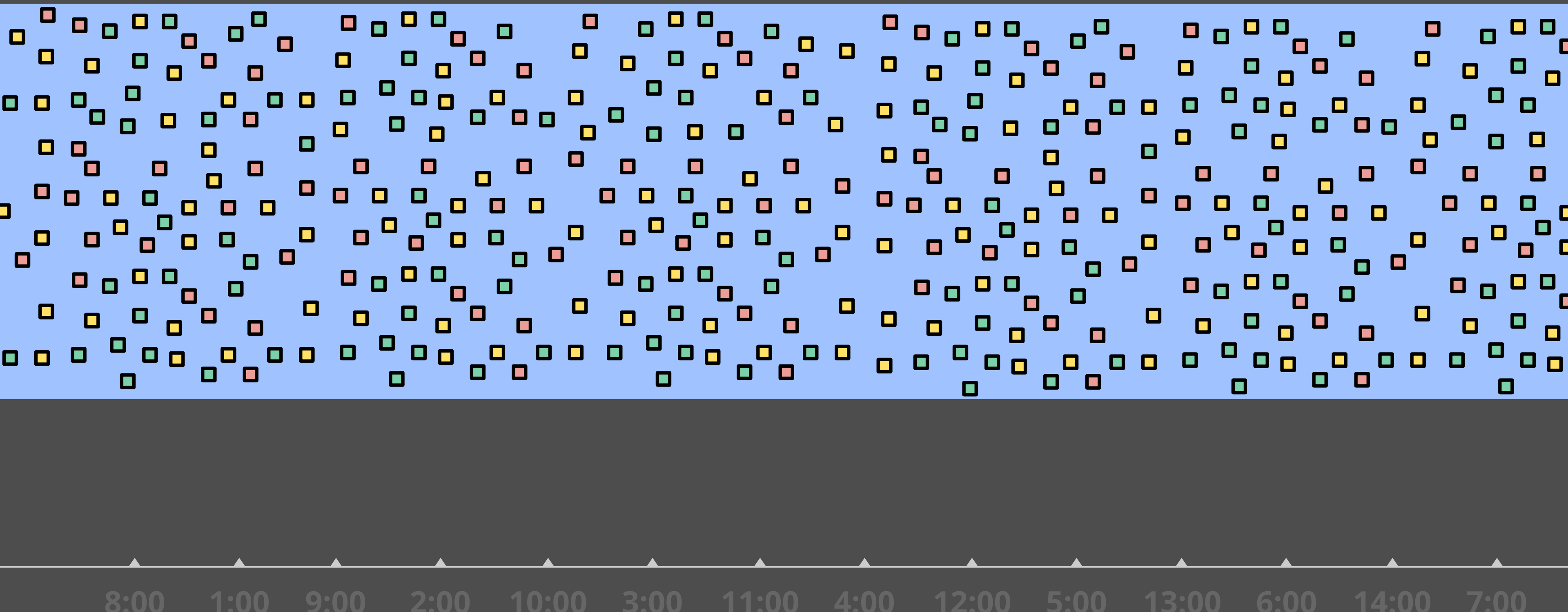
Hourly Ranking

⋮

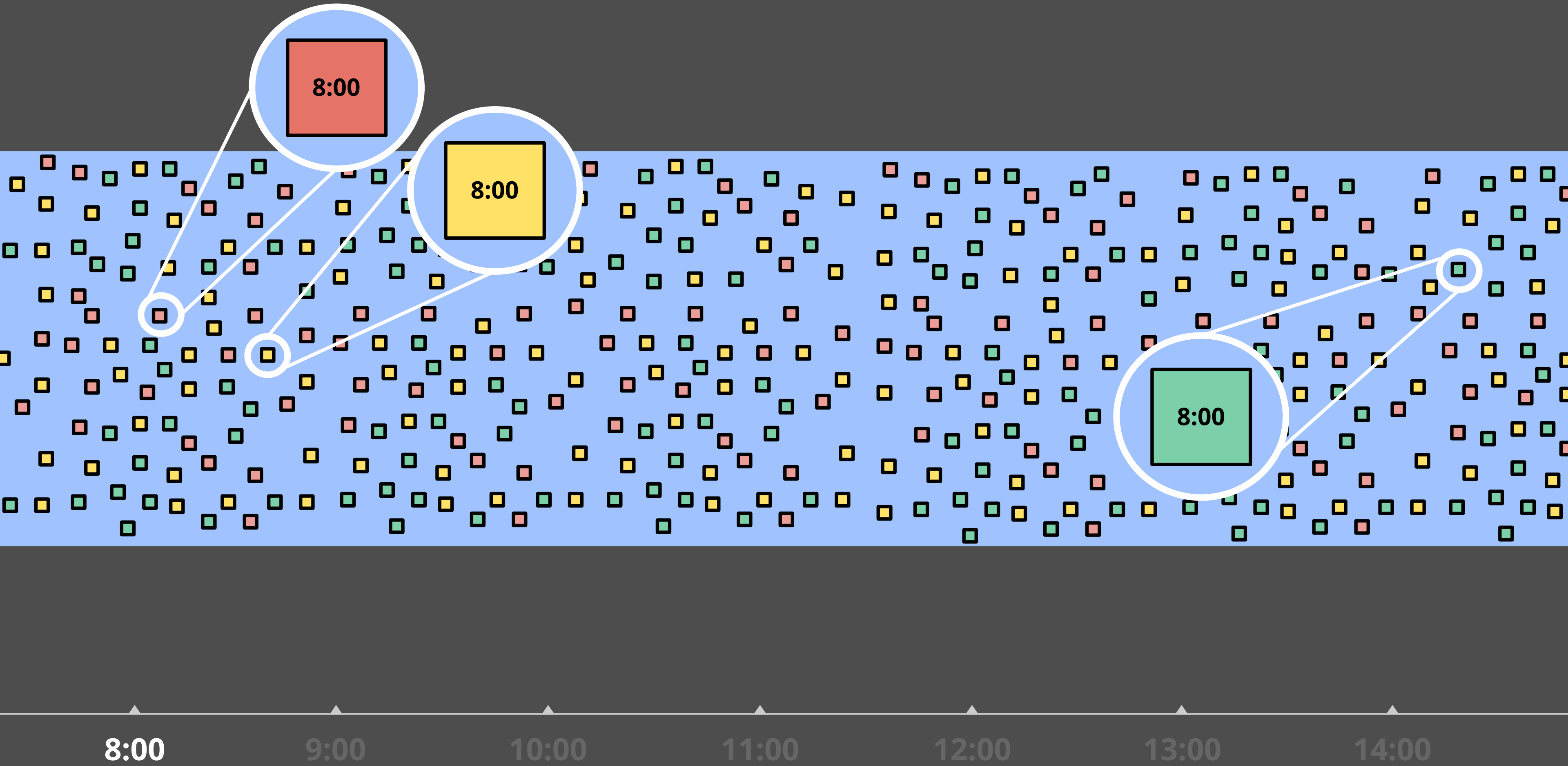
Daily Ranking

⋮

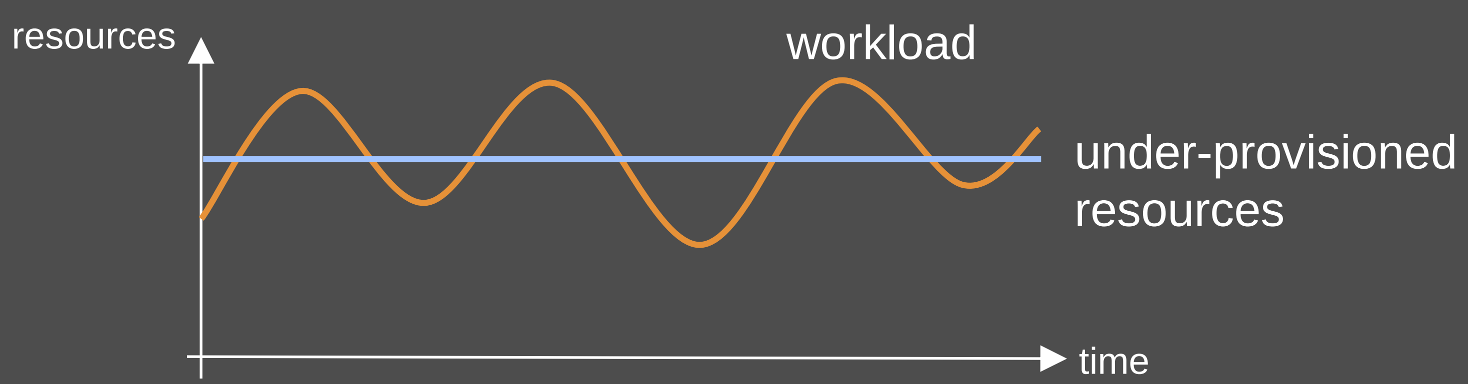
An Unbounded Stream of Game Events



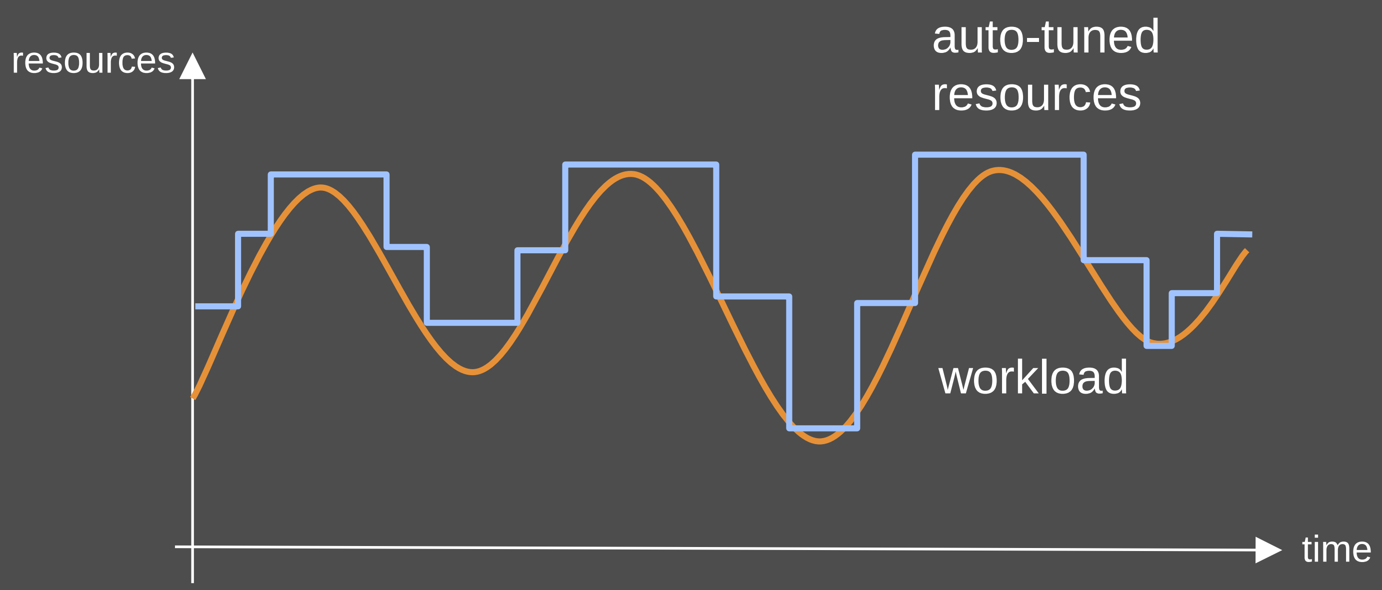
… with unknown delays.



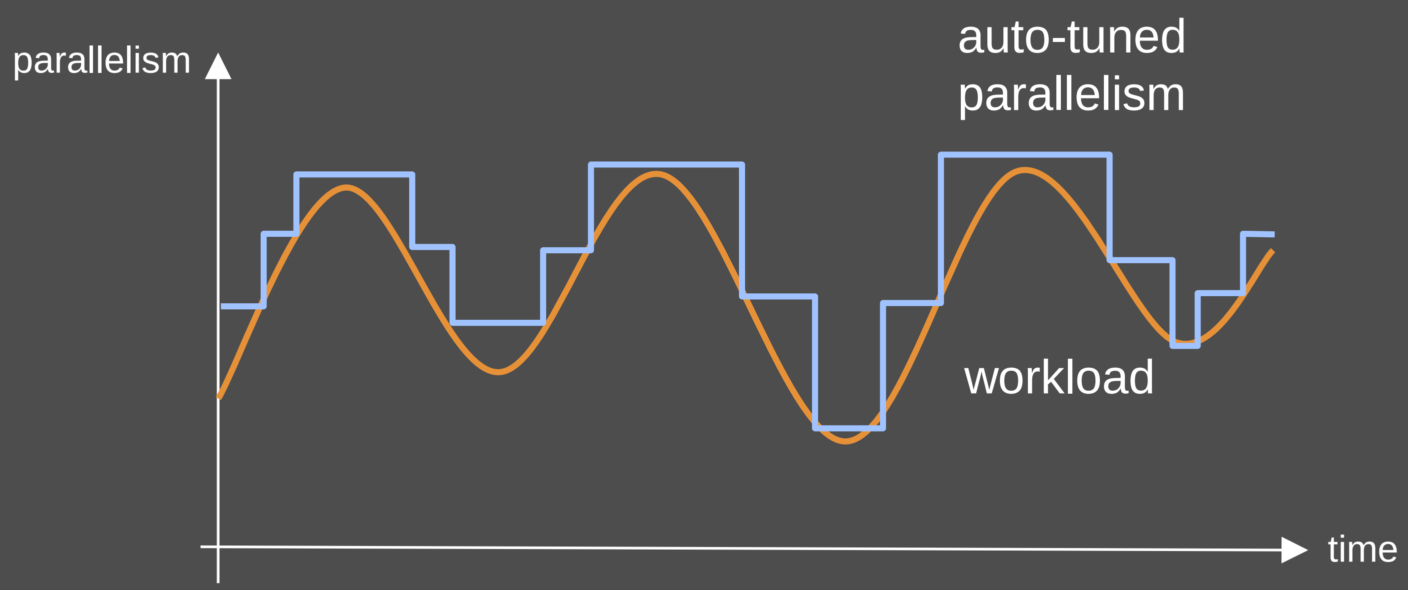
The Resource Allocation Problem



Matching Resources to Workload



Resources = Parallelism



More generally: VMs (including CPU, RAM, network, IO).

Assumptions

Big Data Problem

Embarrassingly Parallel

Scaling VMs ==> Scales Throughput

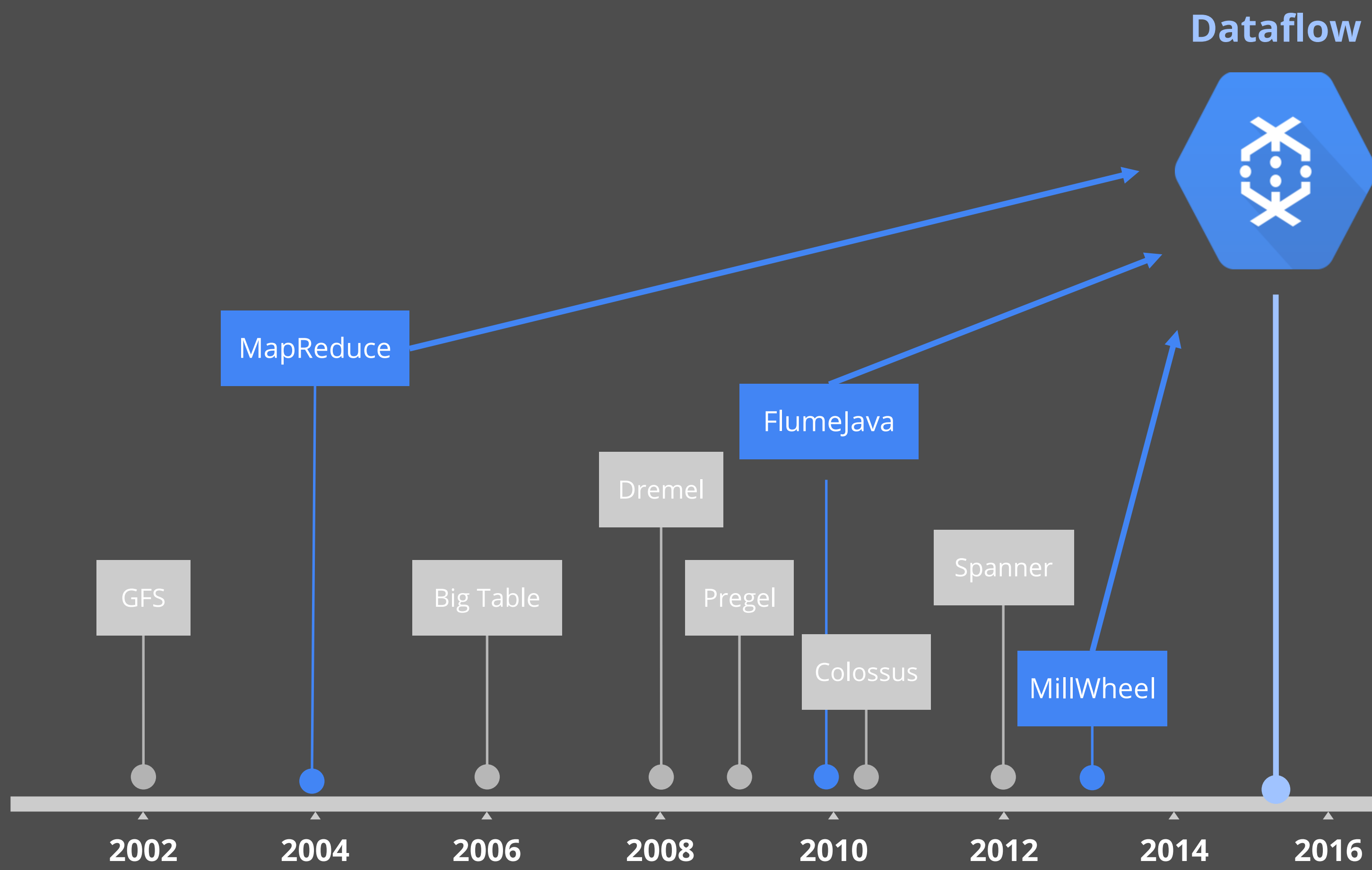
Horizontal Scaling

Agenda

1. Streaming Dataflow Pipelines
2. Pipeline Execution
3. Adjusting Parallelism Automatically
4. Summary + Future Work

1 Streaming Dataflow

Google's Data-Related Systems



Google Dataflow SDK

Open Source SDK used to construct a Dataflow pipeline.

(Now Incubating as Apache Beam)

Computing Team Scores

```
// Collection of raw log lines
PCollection<String> raw = ...;

// Element-wise transformation into team/score
// pairs
PCollection<KV<String, Integer>> input =
    raw.apply(ParDo.of(new ParseFn()))

// Composite transformation containing an
// aggregation
PCollection<KV<String, Integer>> output = input
  .apply(Window.into(FixedWindows.of(Minutes(60))))
  .apply(Sum.integersPerKey());
```

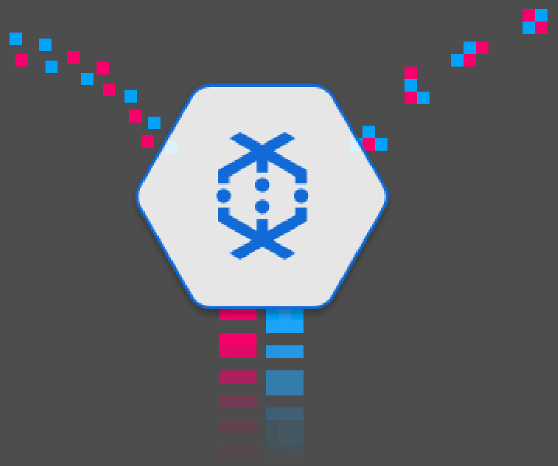
Google Cloud Dataflow

- Given code in Dataflow (incubating as Apache Beam) SDK...

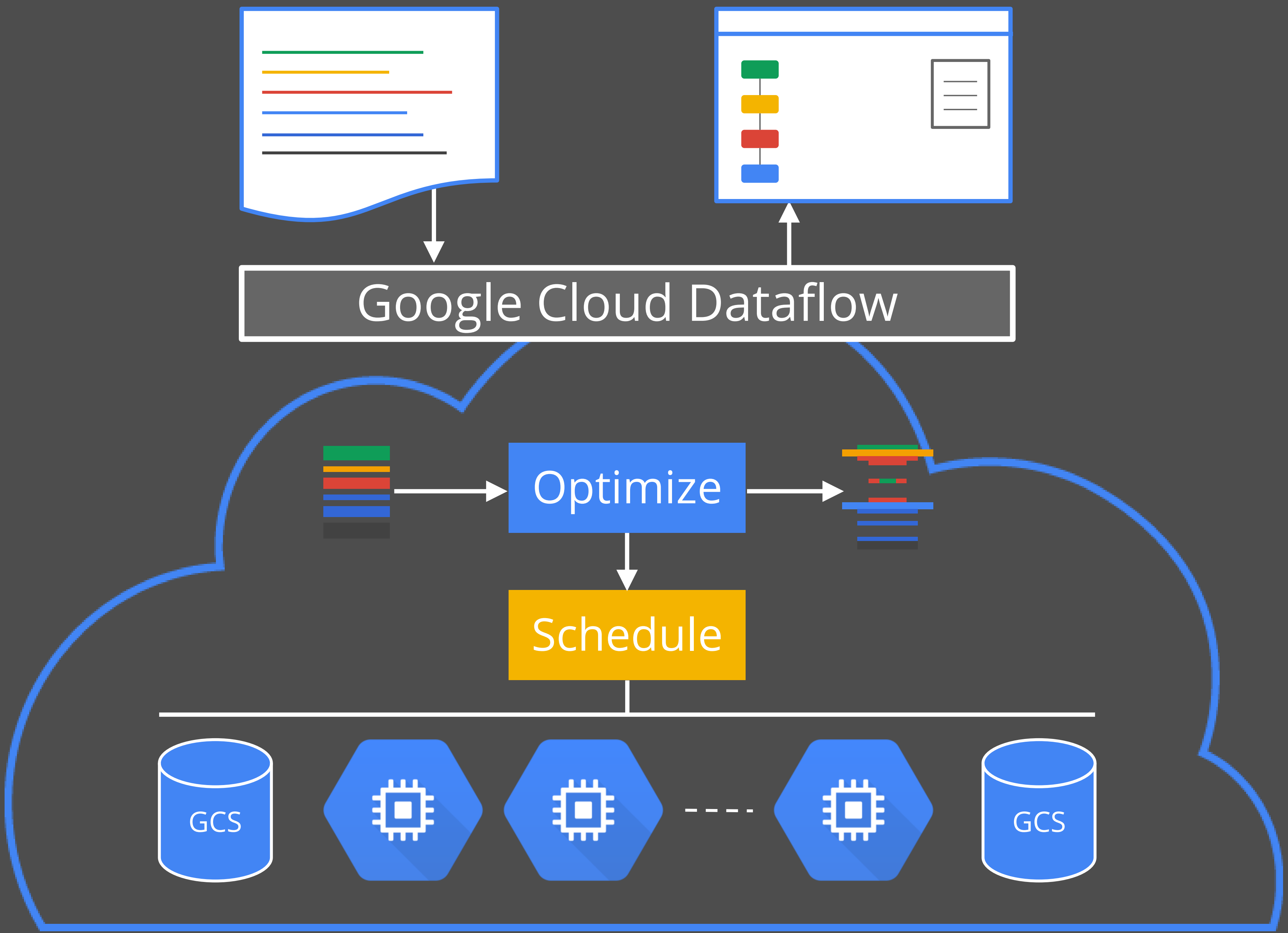
- Pipelines can run…
  - On your development machine
  - On the Dataflow Service on Google Cloud Platform
  - On third party environments like Spark or Flink.

Google Cloud Dataflow

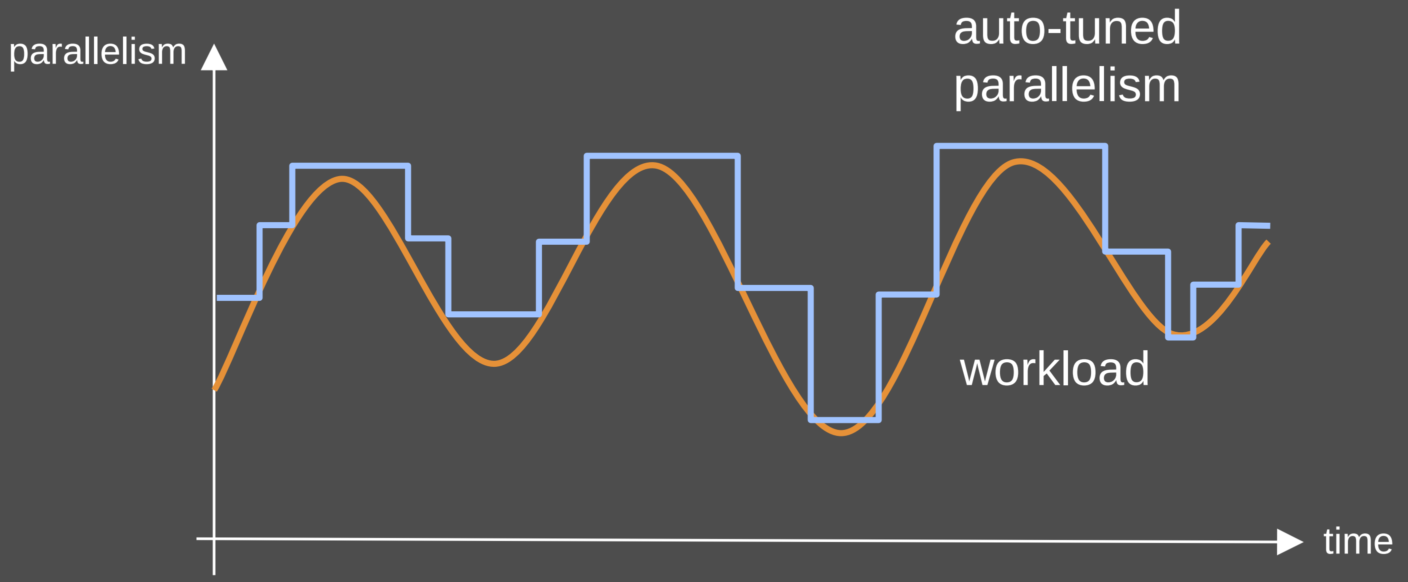
Google Cloud Platform



A fully-managed cloud service and programming model for batch and streaming big data processing.


Google Cloud Dataflow
Optimize
Schedule
GCS
GCS

Back to the Problem at Hand



Auto-Tuning Ingredients

Signals measuring Workload

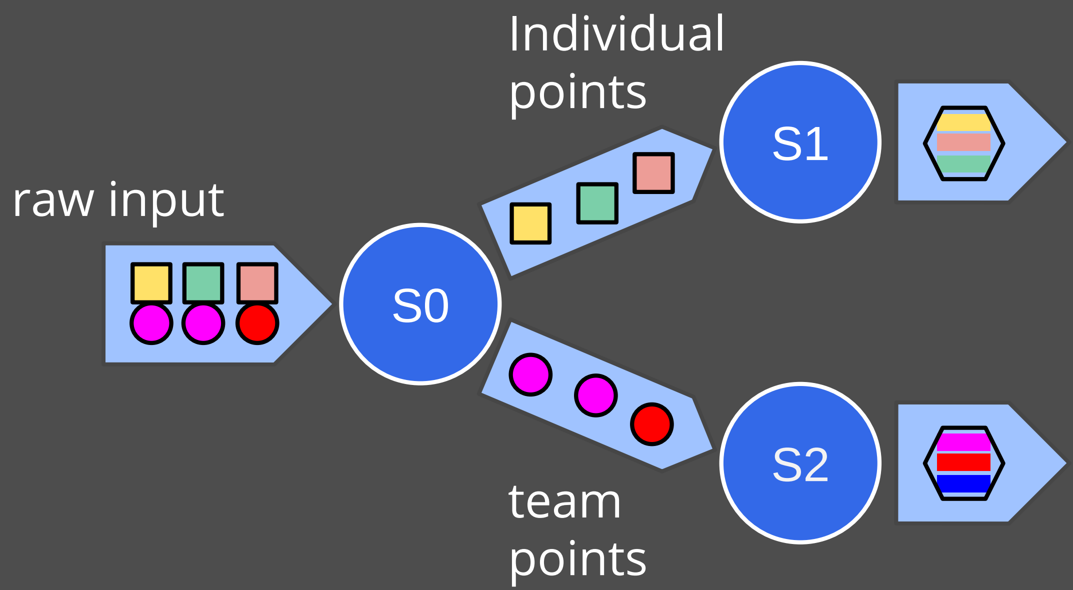
Policy making Decisions

Mechanism actuating Change

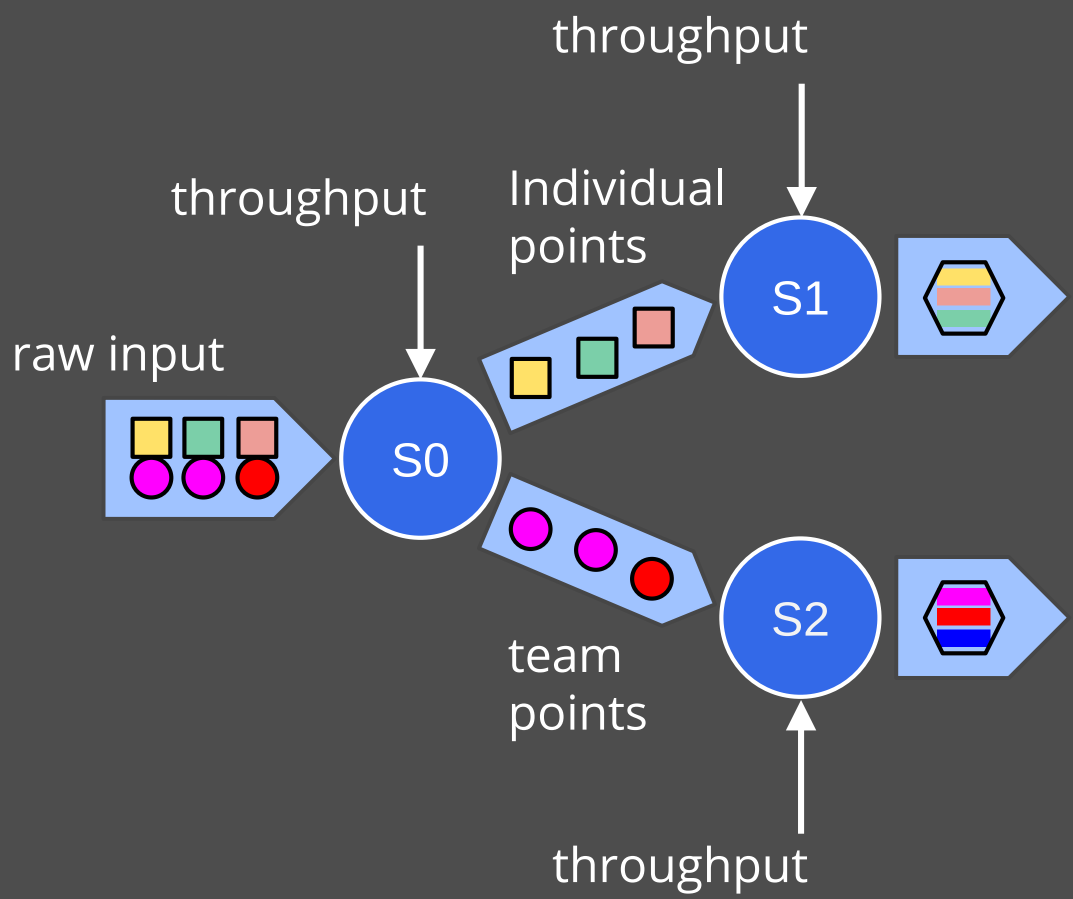
2

Pipeline Execution

Optimized Pipeline = DAG of Stages

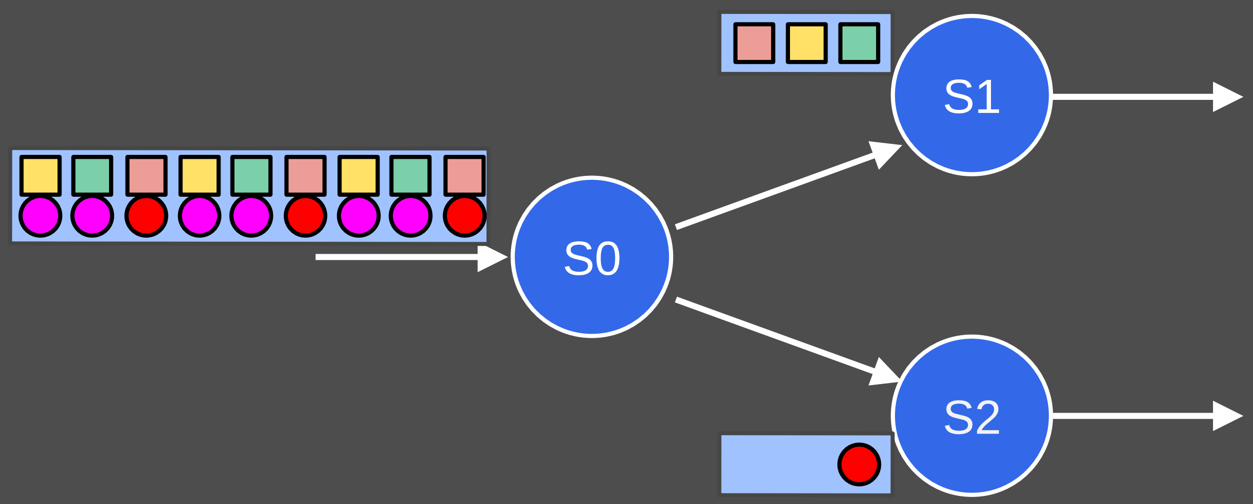


Stage Throughput Measure



Picture by Alexandre Duret-Lutz, Creative Commons 2.0 Generic

Queues of Data Ready for Processing



Queue Size = Backlog

Backlog Size

vs.

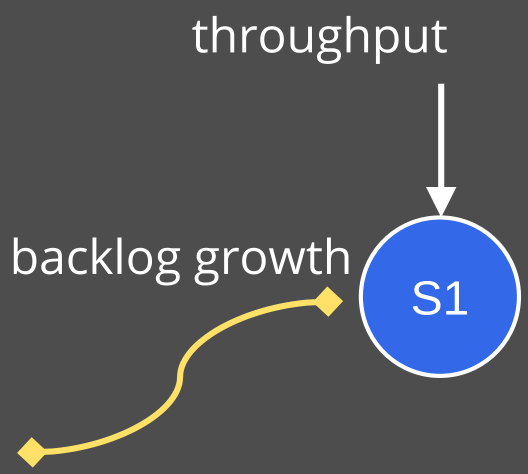
Backlog Growth

Backlog Growth

=

Processing Deficit

Derived Signal: Stage Input Rate



Input Rate = Throughput + Backlog Growth

Constant Backlog…

…could be bad

$$\text{Backlog Time} = \frac{\text{Backlog Size}}{\text{Throughput}}$$

Backlog Time =

Time to get through backlog

Bad Backlog = Long Backlog Time

Backlog Growth
and
Backlog Time
Inform Upscaling.

What Signals indicate
Downscaling?

Low CPU Utilization

Signals Summary

Throughput

Backlog growth

Backlog time

CPU utilization

Policy: making Decisions

Goals:

1. No backlog growth
2. Short backlog time
3. Reasonable CPU utilization

Upscaling Policy: Keeping Up

Given M machines

For a stage, given:

average stage throughput T

average positive backlog growth G of stage

Machines needed for stage to keep up:

$$M' = M \frac{(T + G)}{T}$$

Upscaling Policy: Catching Up

Given M machines

Given R (time to reduce backlog)

For a stage, given:

average backlog time B

Extra machines to remove backlog:

$$Extra = M \frac{B}{R}$$

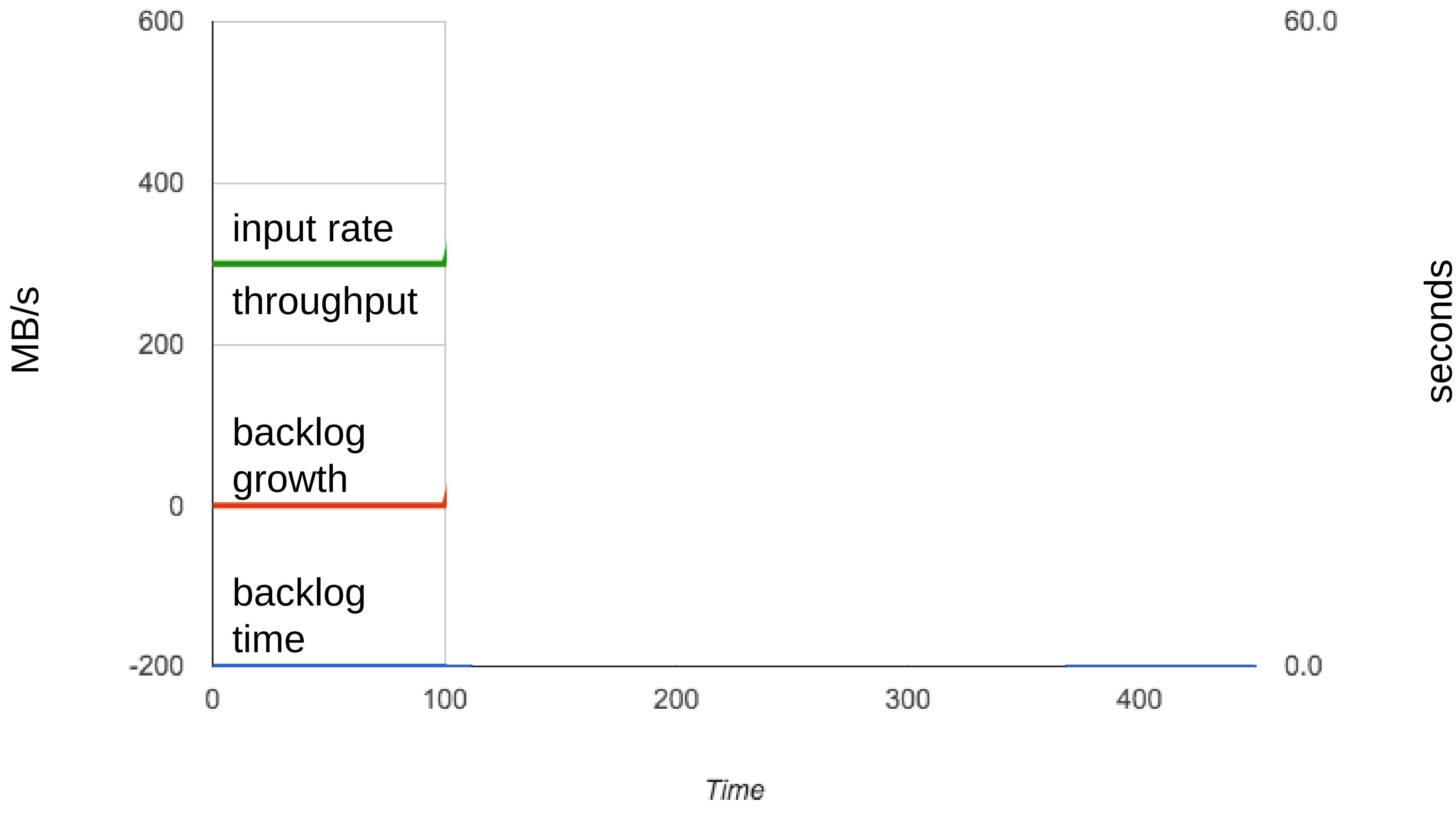
Upscaling Policy: All Stages

Want all stages to:

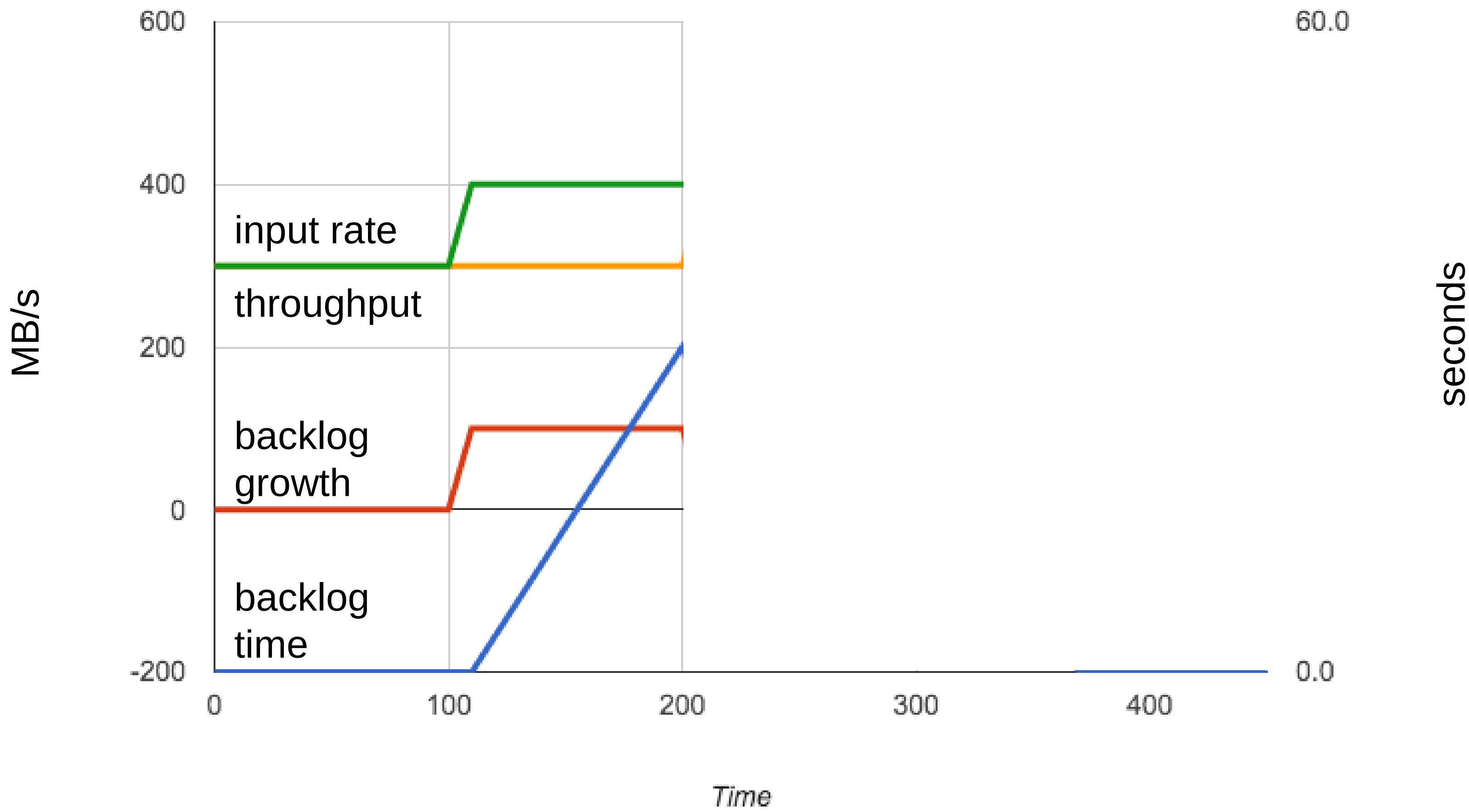
1. keep up
2. have log backlog time

Pick Maximum over all stages of $M' + Extra$

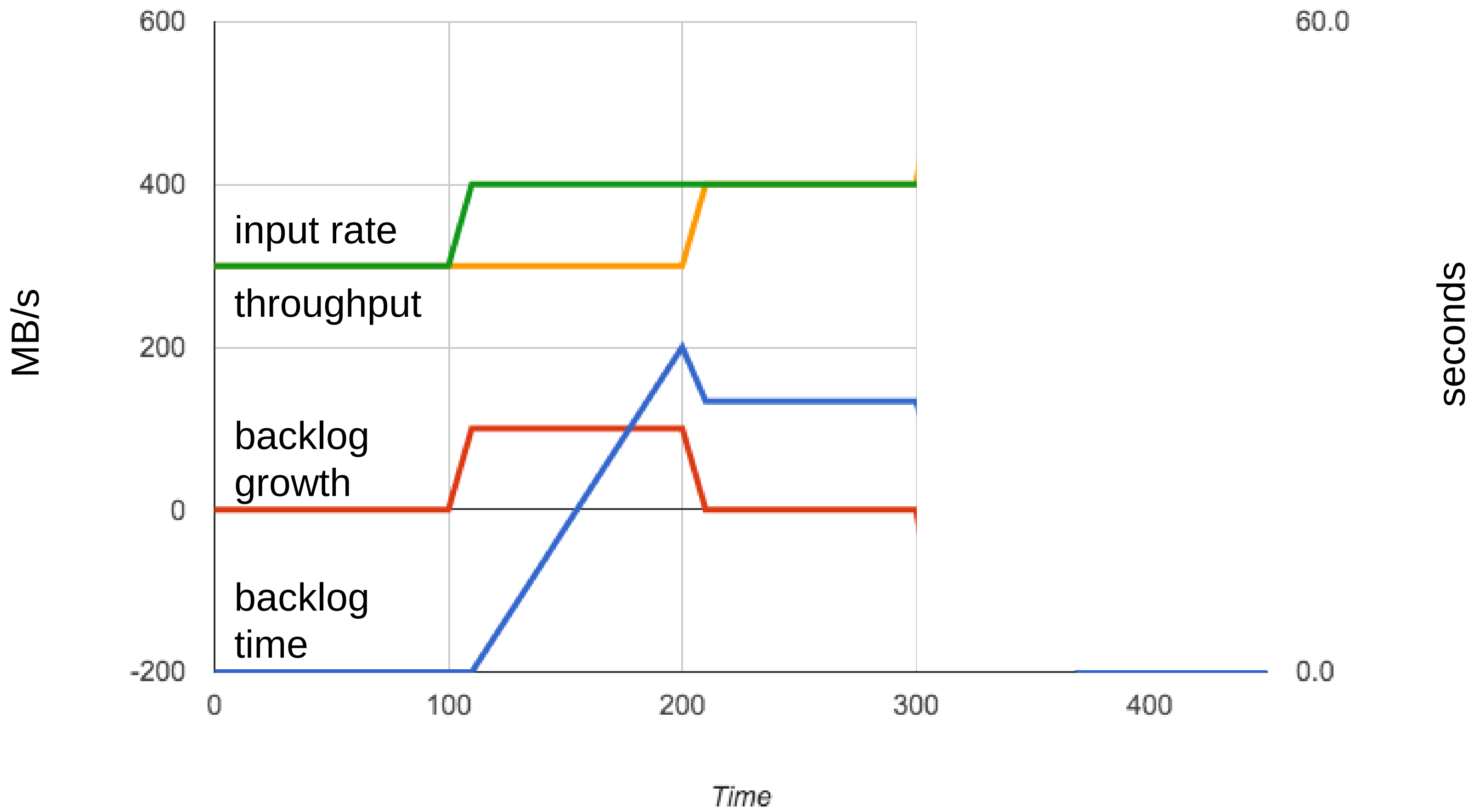
Example (signals)



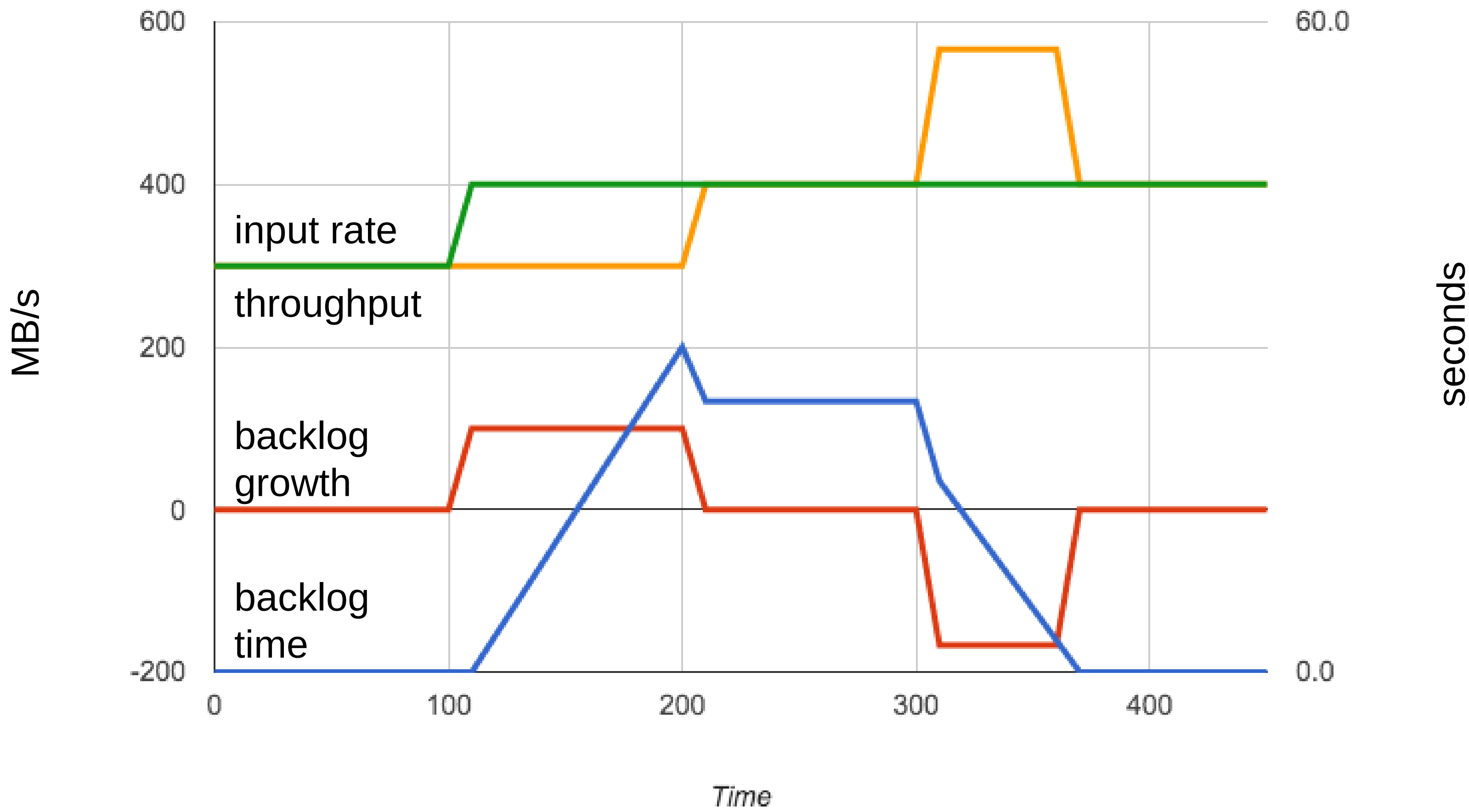
Example (signals)



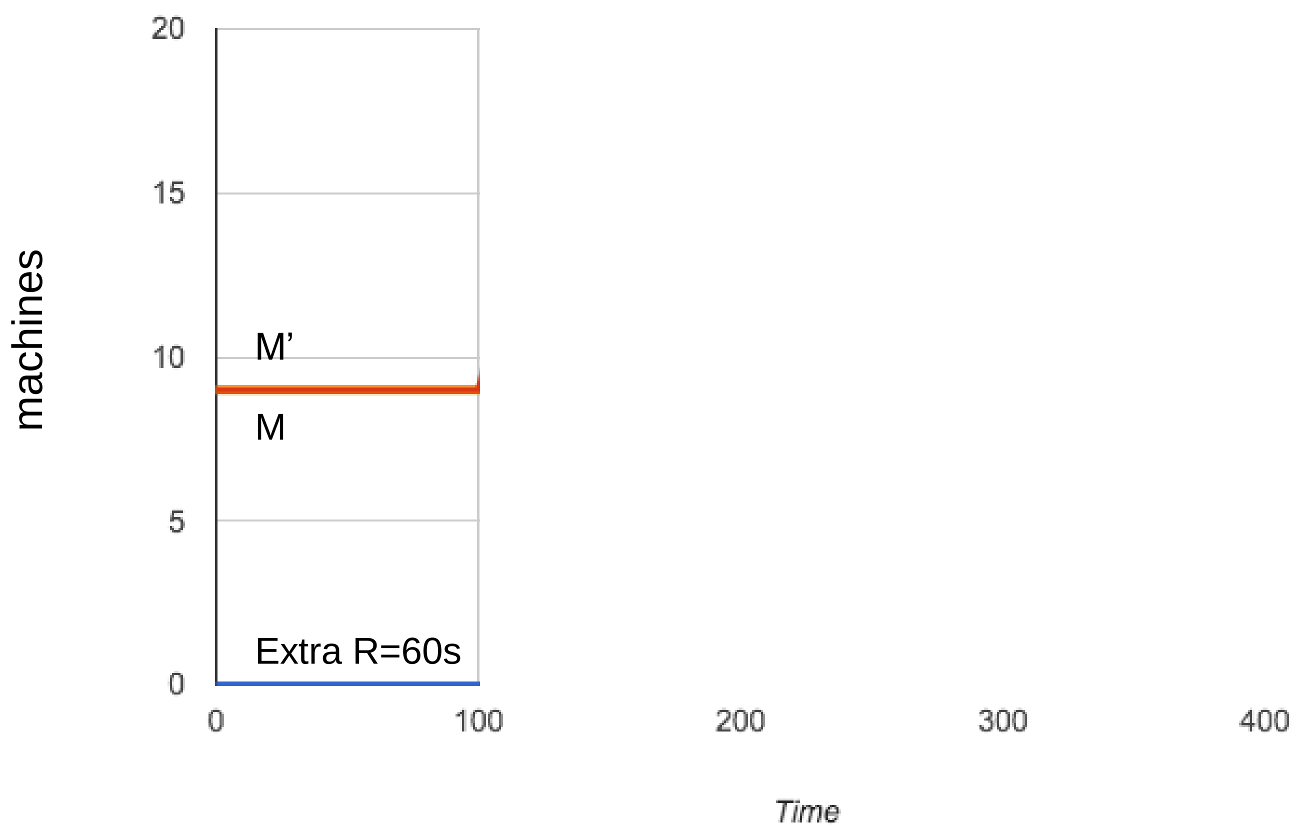
Example (signals)



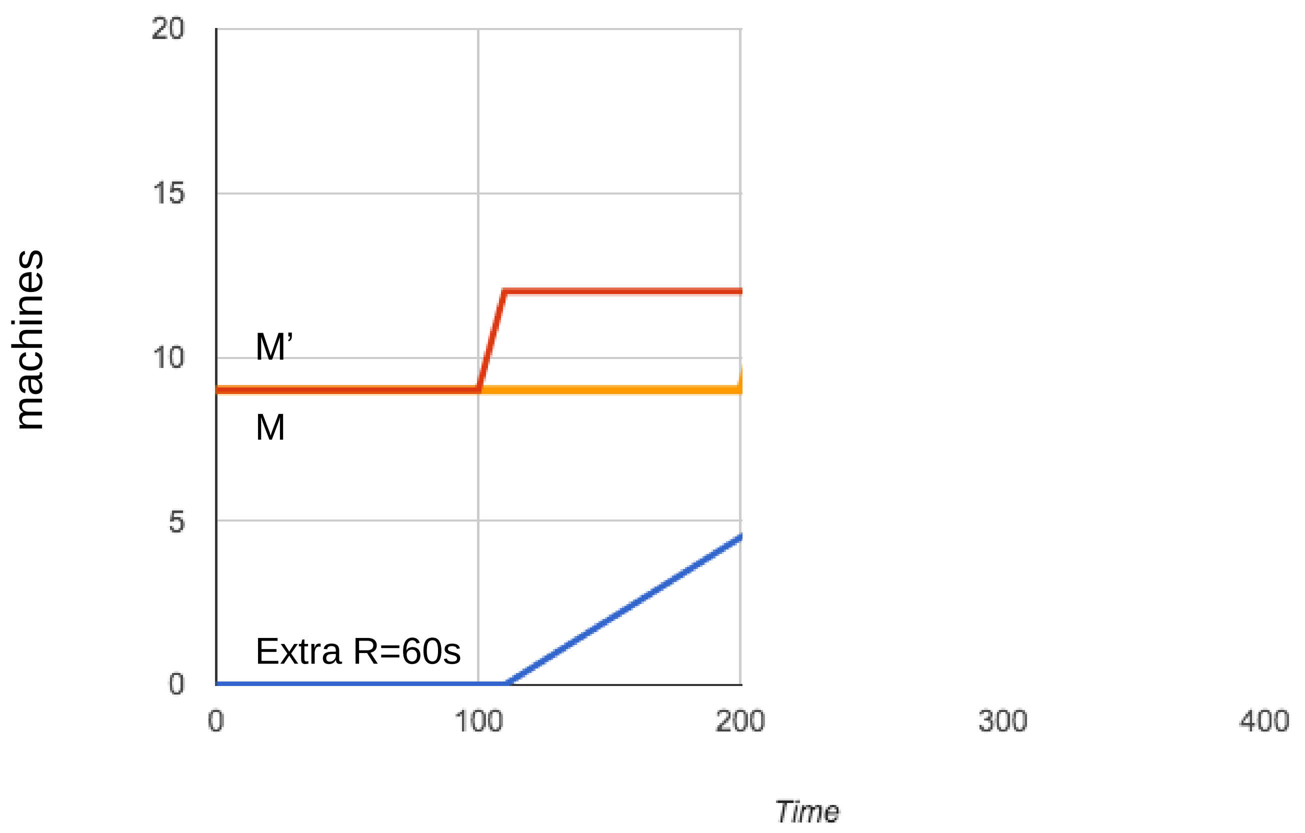
Example (signals)



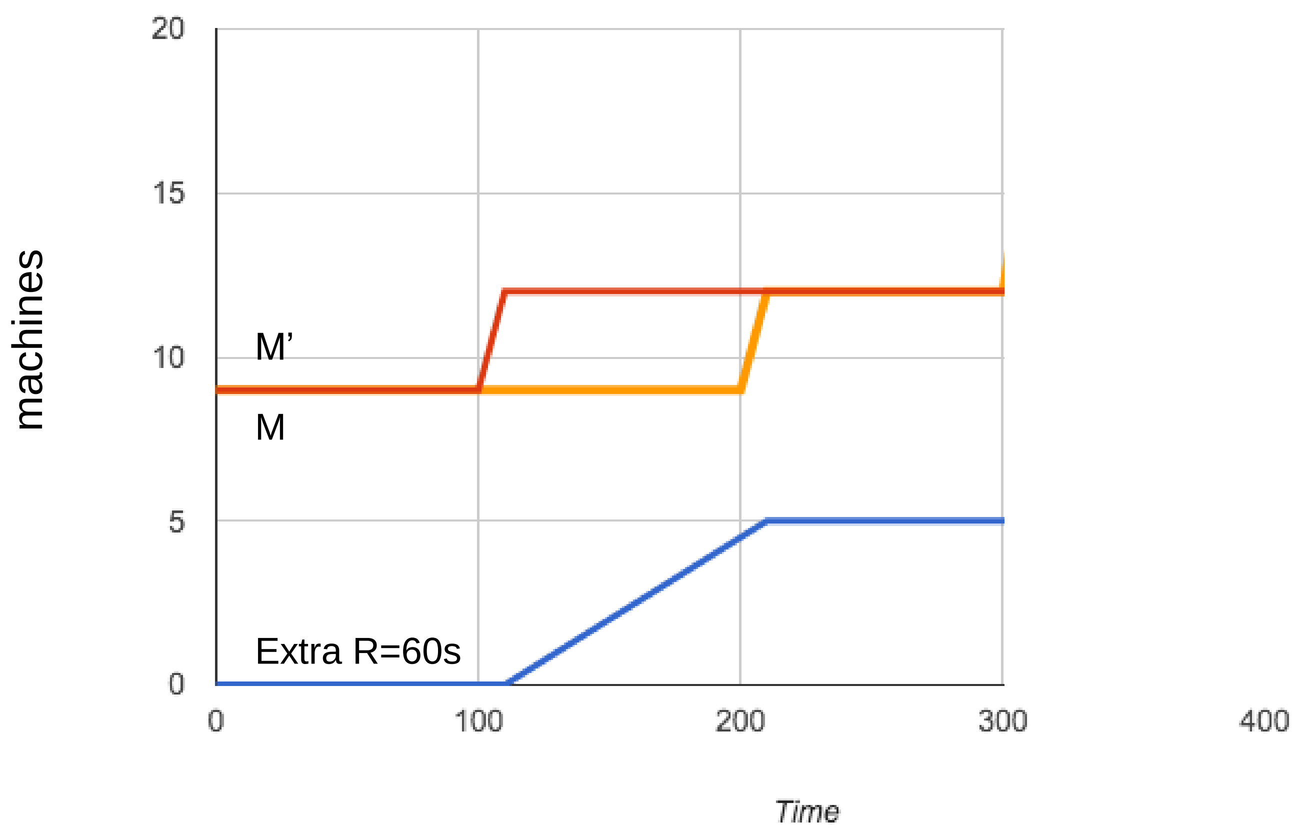
Example (policy)



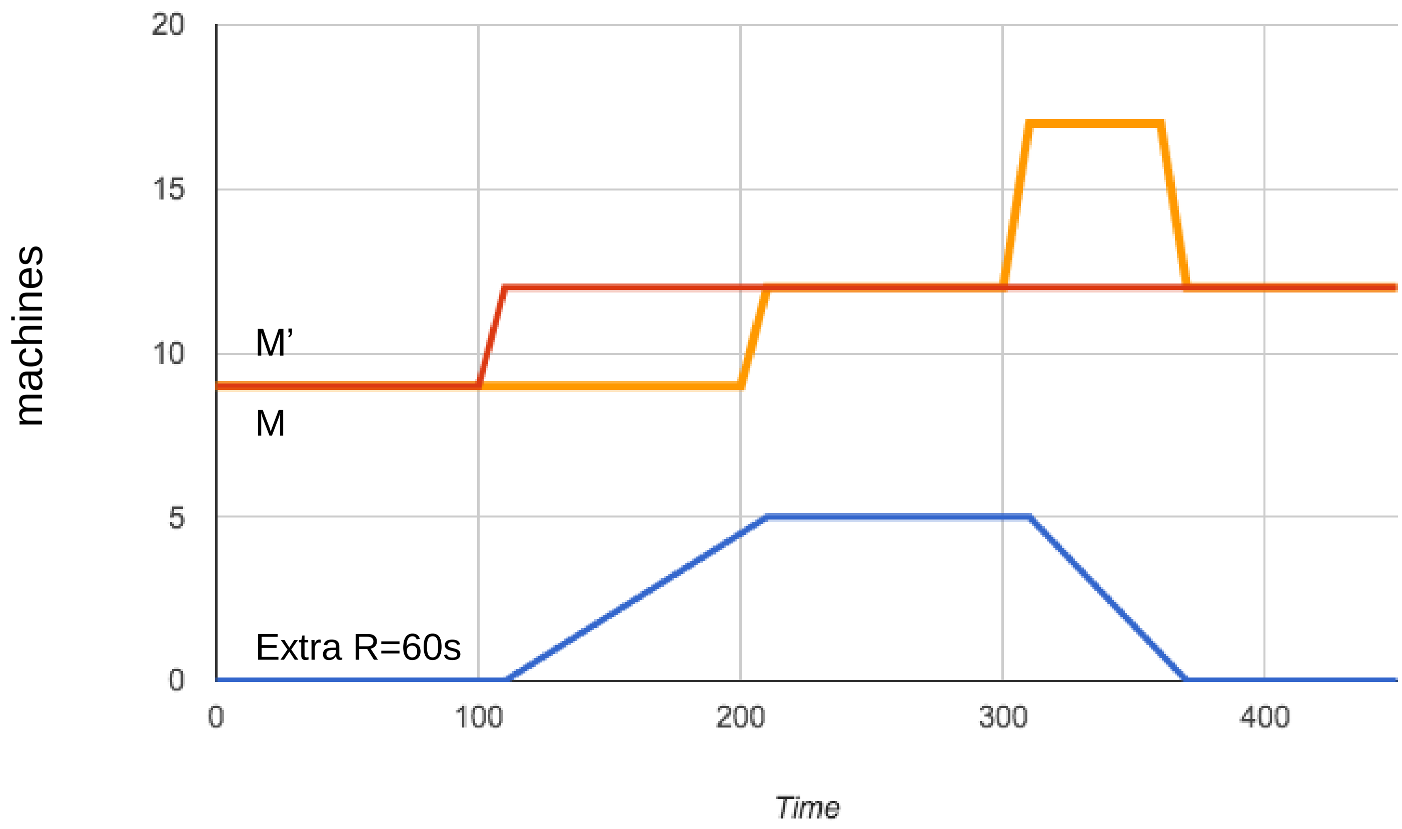
Example (policy)



Example (policy)



Example (policy)



Preconditions for Downscaling

Low backlog time

No backlog growth

Low CPU utilization

How far can we downscale?

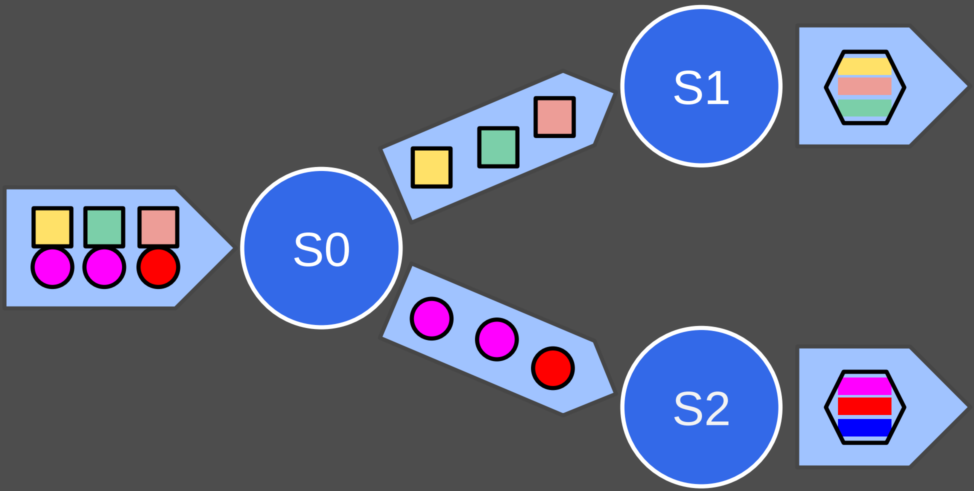
Stay tuned...

Mechanism: actuating Change

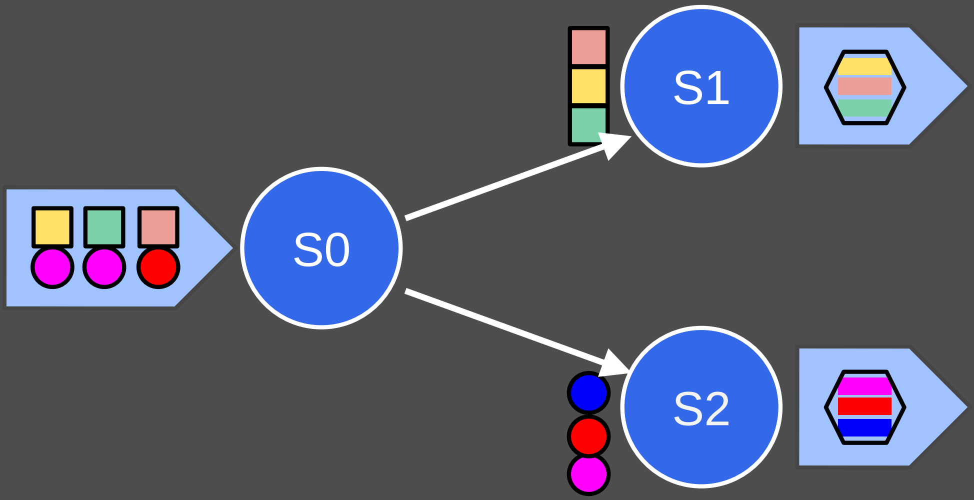
3

Adjusting Parallelism of a Running Streaming Pipeline

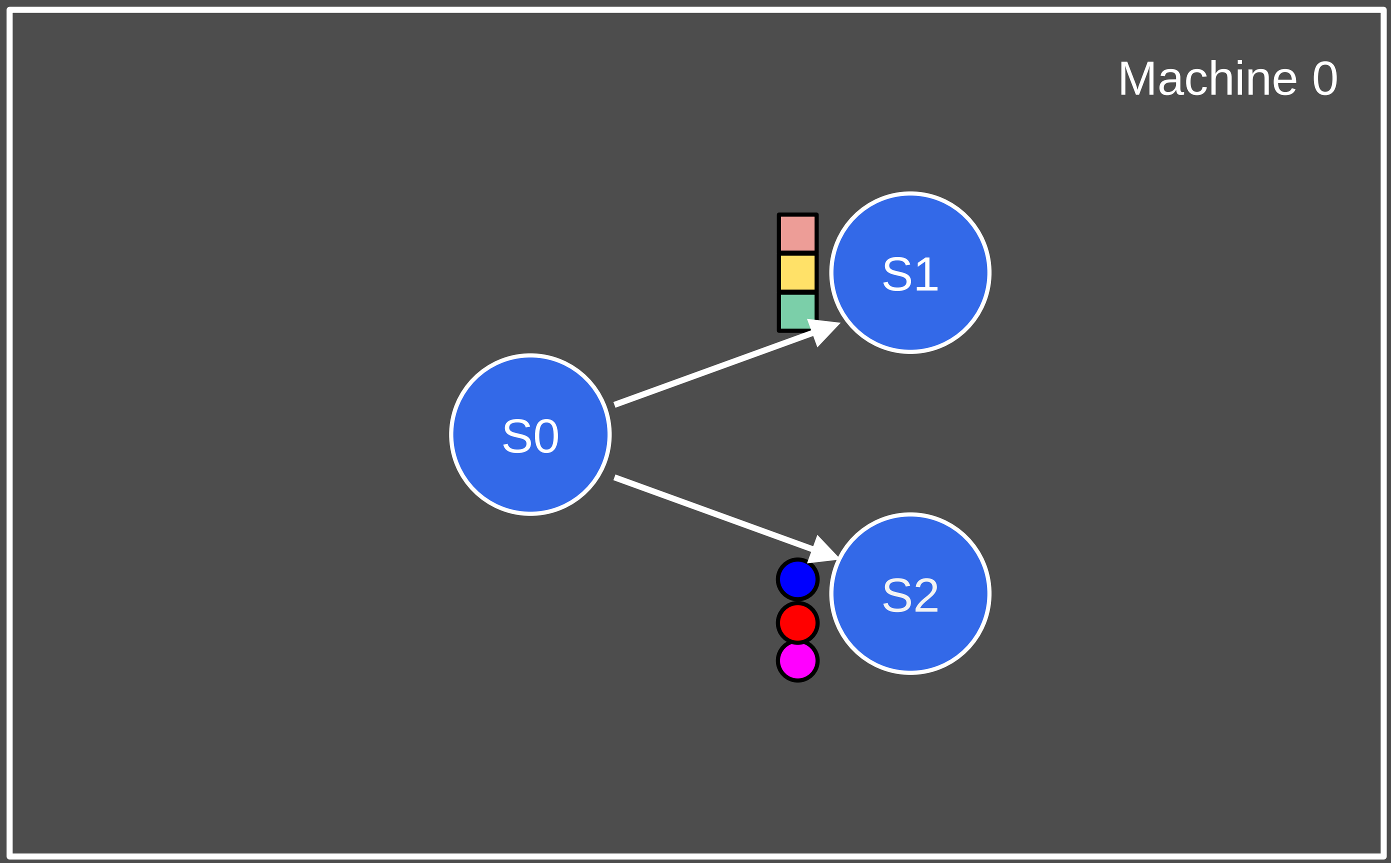
Optimized Pipeline = DAG of Stages



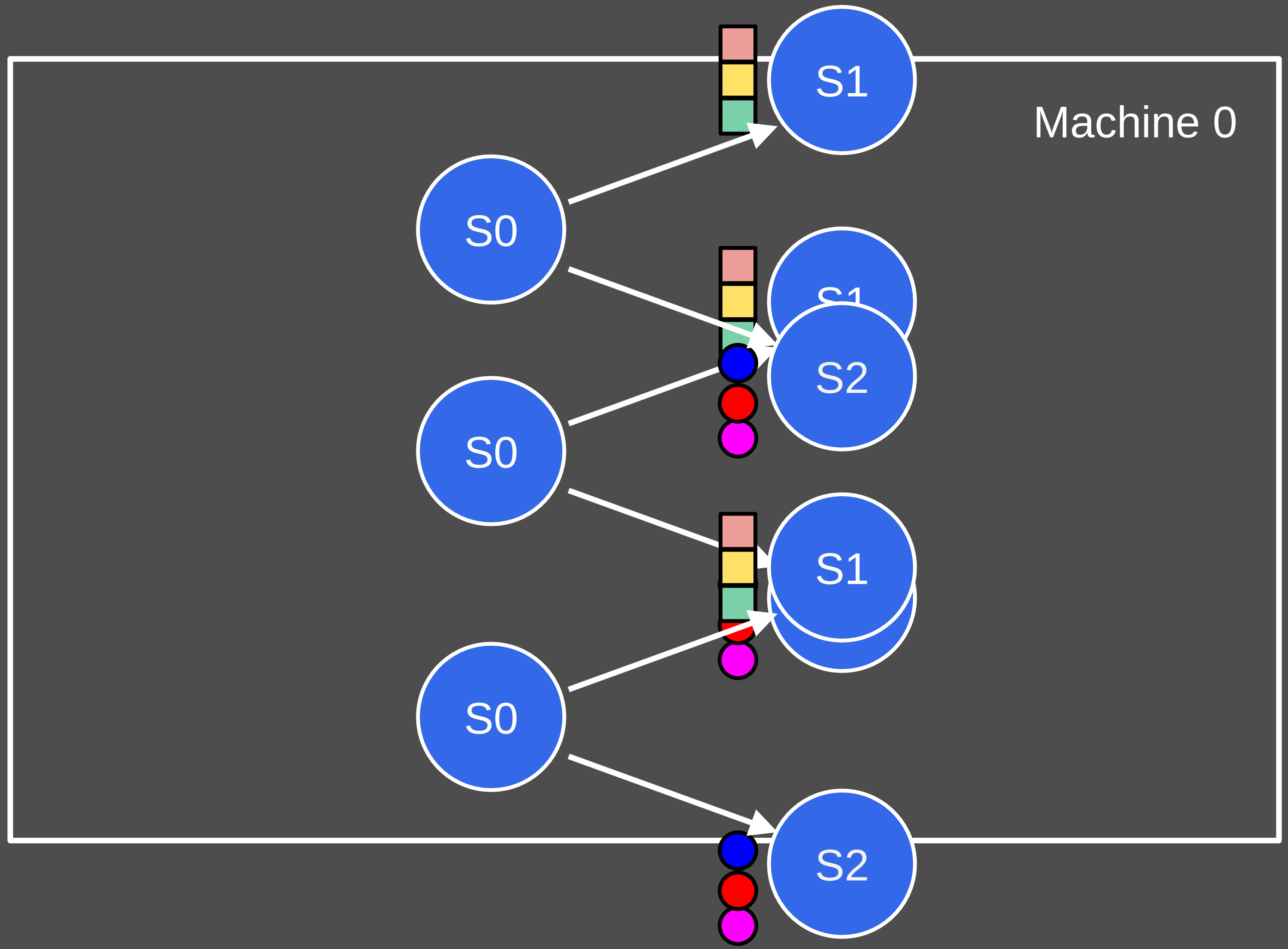
Optimized Pipeline = DAG of Stages



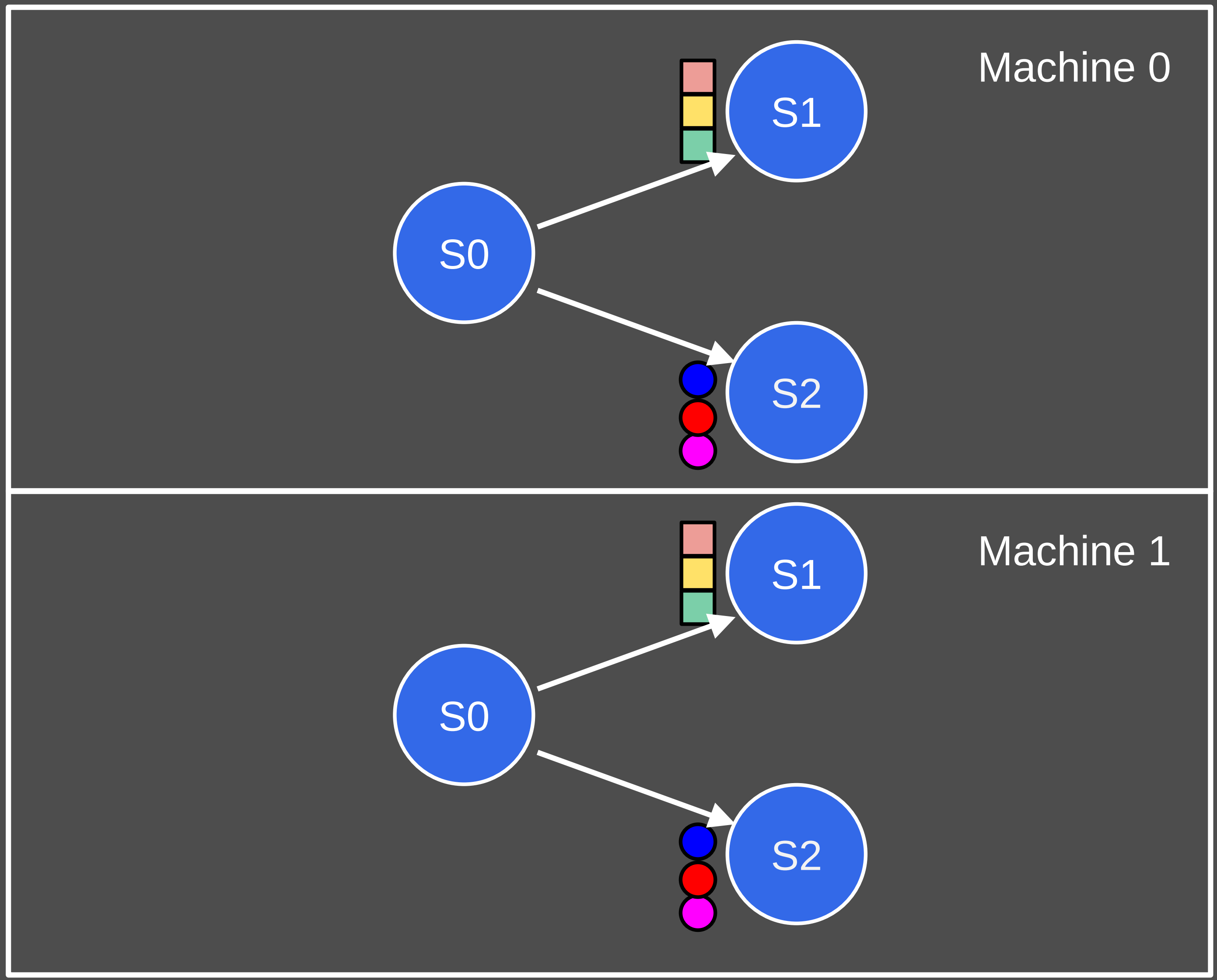
Optimized Pipeline = DAG of Stages



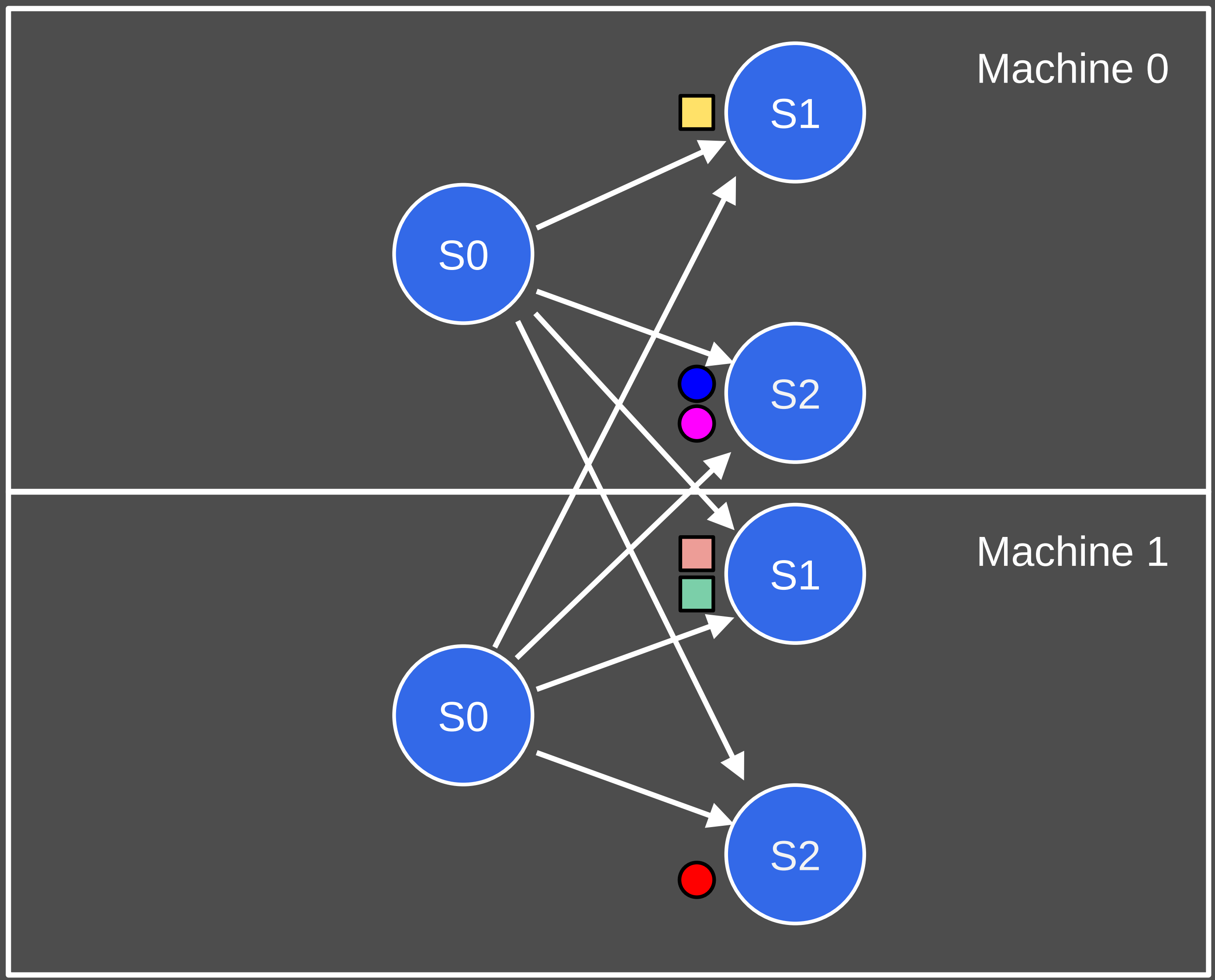
Adding Parallelism



Adding Parallelism

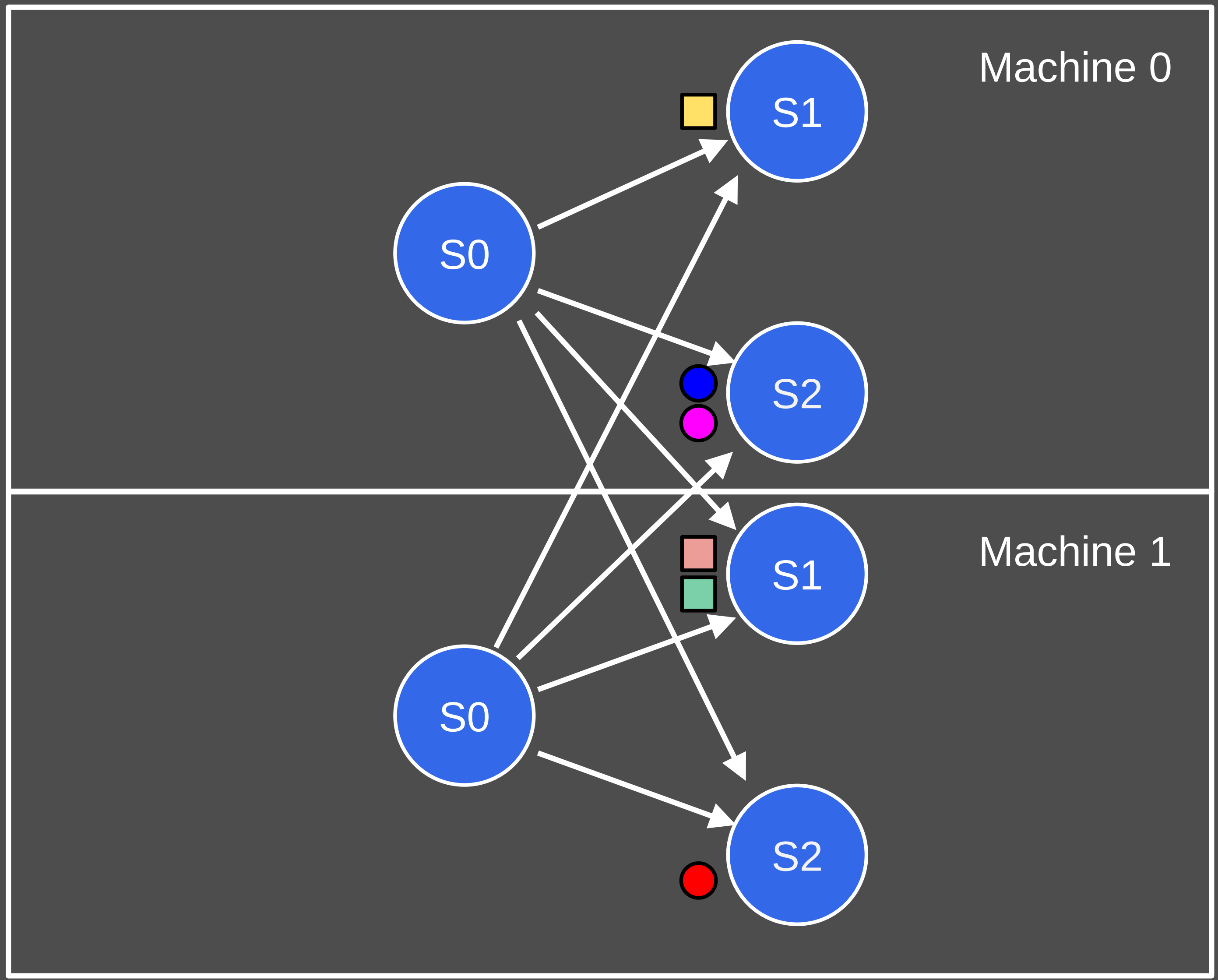


Adding Parallelism = Splitting Key Ranges



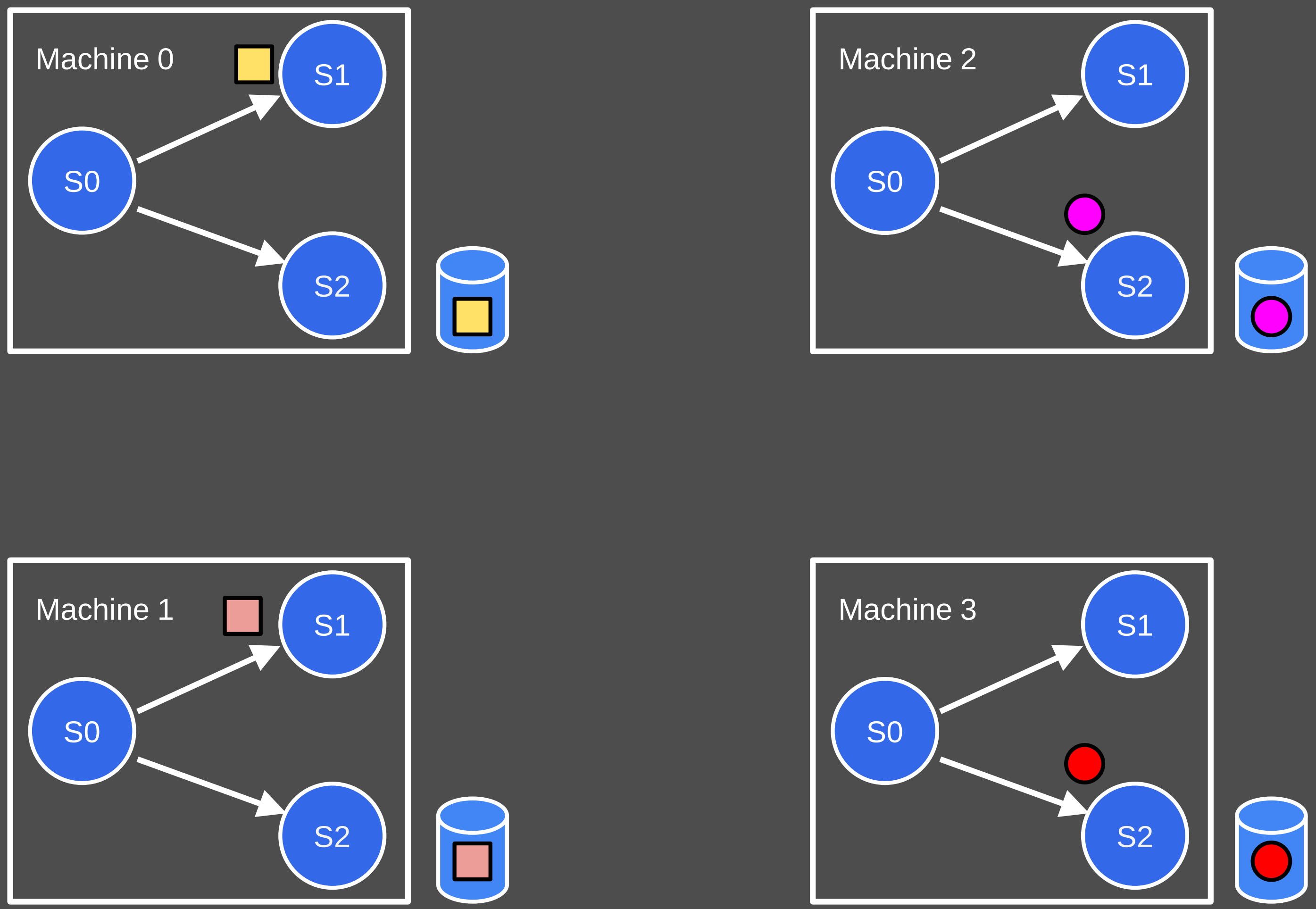
Migrating a Computation

Adding Parallelism = Migrating Computation Ranges

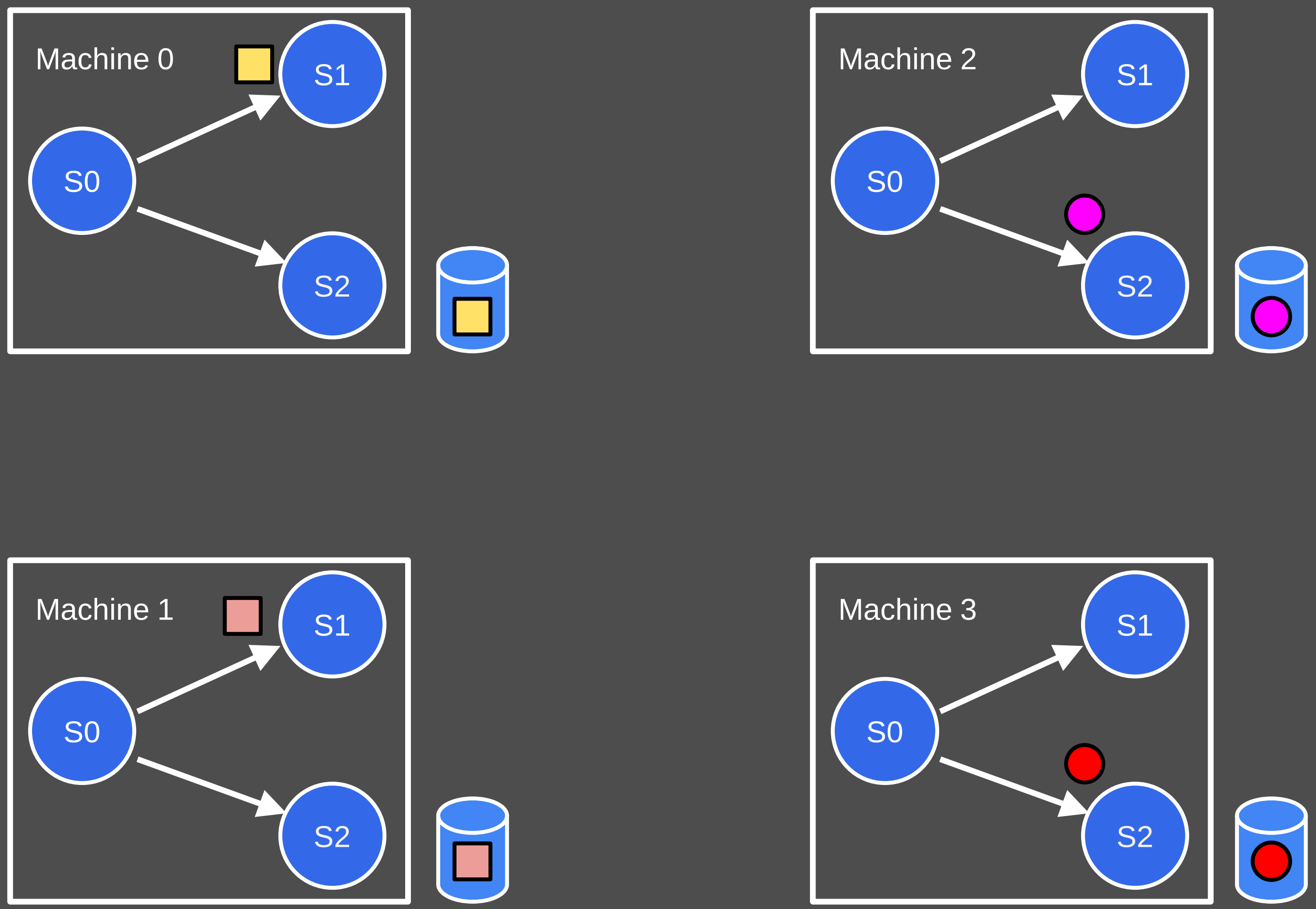


Checkpoint and Recovery
~
Computation Migration

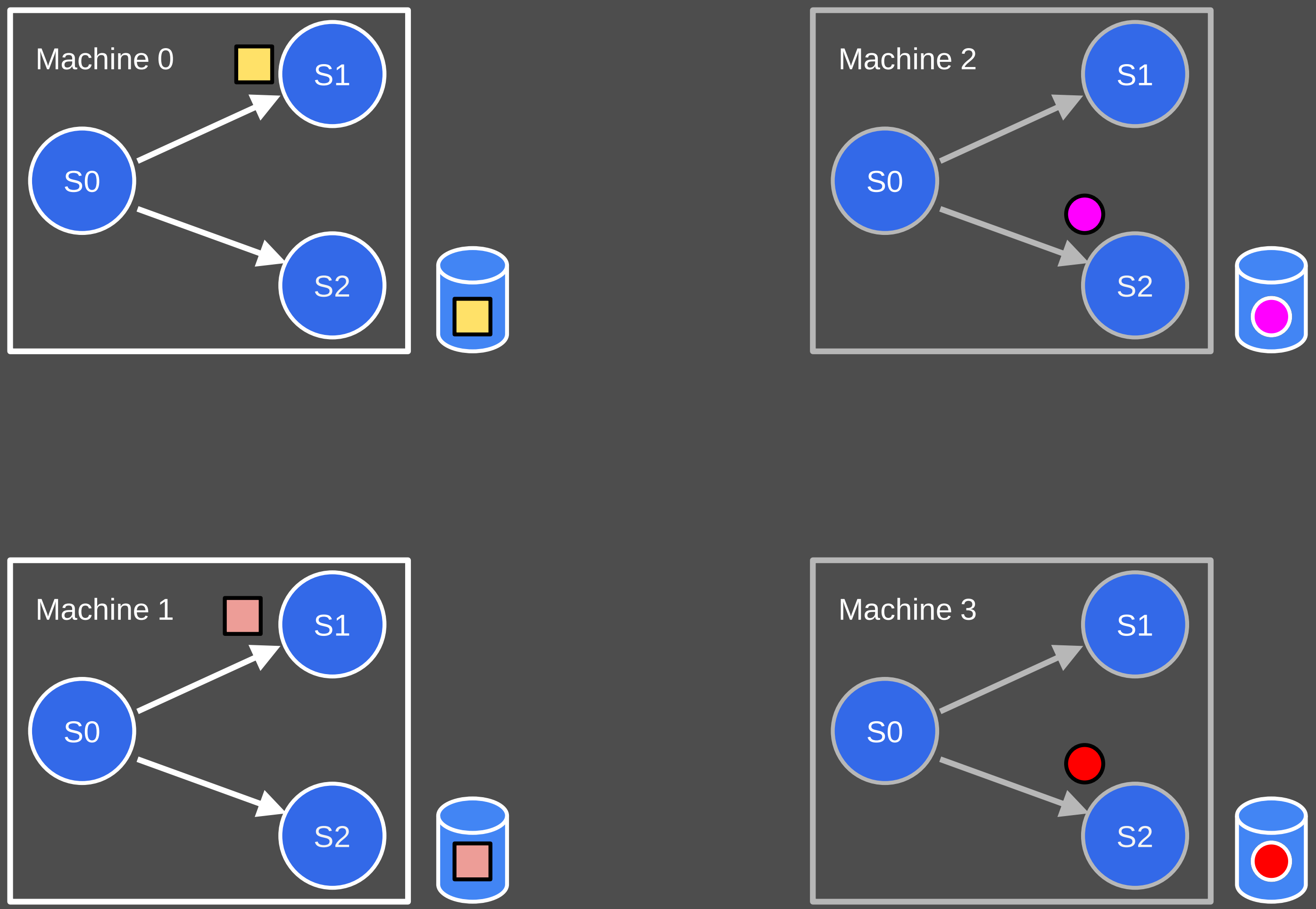
Key Ranges and Persistence



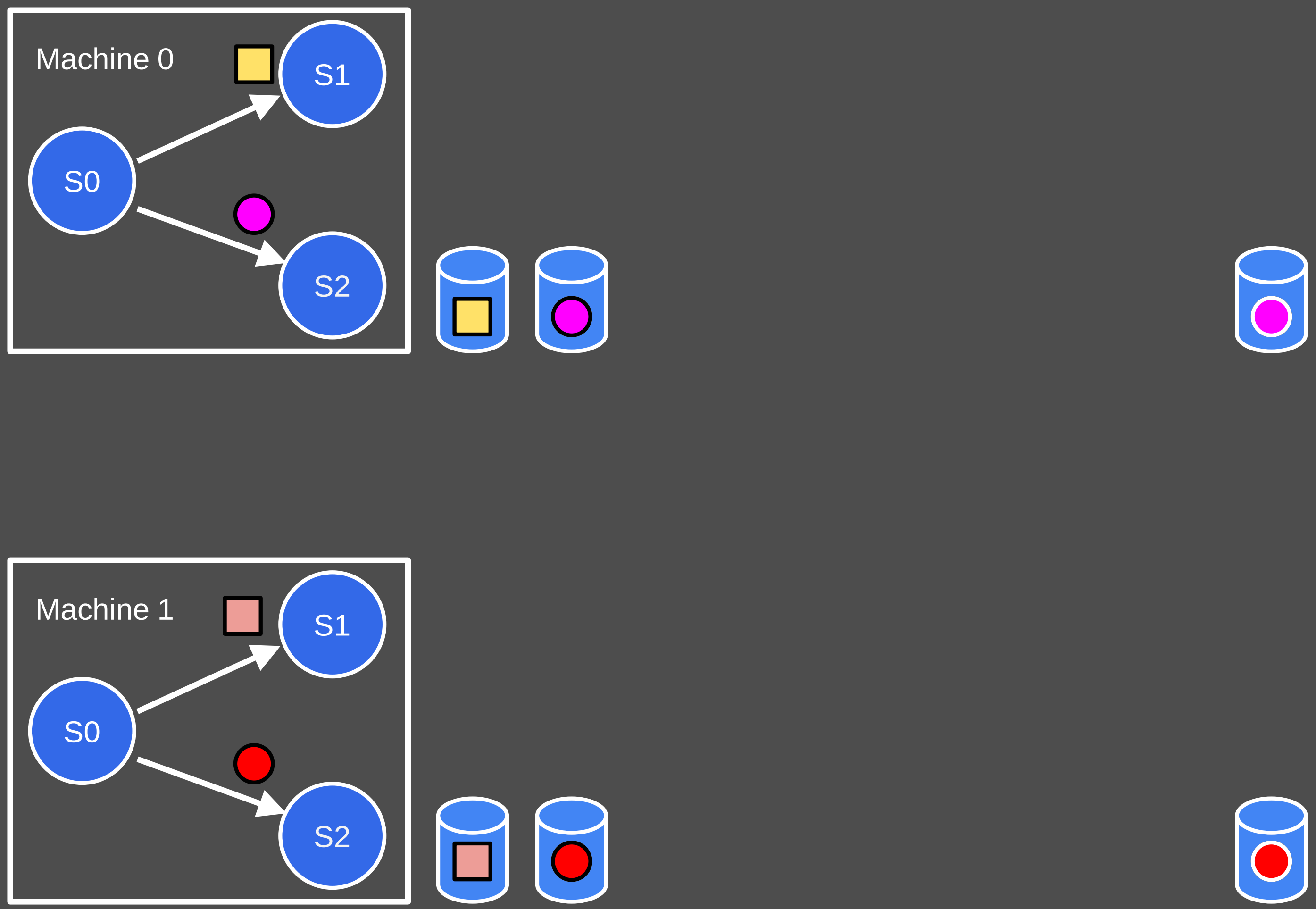
Downscaling from 4 to 2 Machines



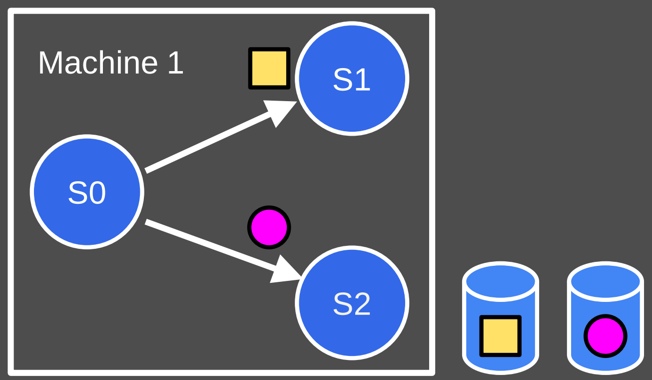
Downscaling from 4 to 2 Machines



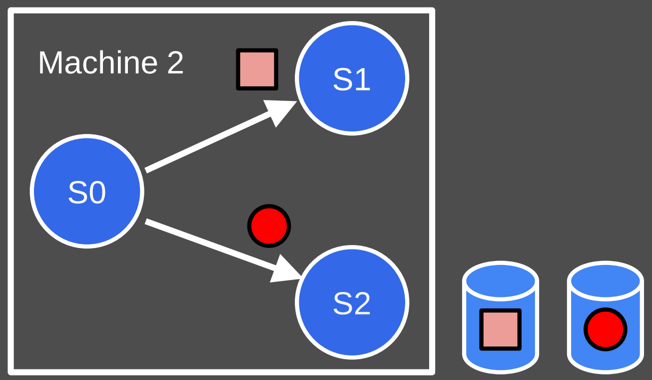
Downscaling from 4 to 2 Machines



Downscaling from 4 to 2 Machines



Upsizing = Steps in Reverse



Granularity of Parallelism

As of March 2016, Google Cloud Dataflow:

- Splits Key Ranges initially Based on Max Machines
- At Max: 1 Logical Persistent Disk per Machine
  Each disk has slice of key ranges from all stages
- Only (relatively) even Disk Distributions
- Results in Scaling Quanta

Example Scaling Quanta:
Max = 60 Machines

Parallelism	Disk per Machine
3	N/A
4	15
5	12
6	10
7	8, 9
8	7, 8
9	6, 7
10	6
12	5
15	4
20	3
30	2
60	1

Policy: making Decisions

Goals:

1. No backlog growth
2. Short backlog time
3. Reasonable CPU utilization

Preconditions for Downscaling

Low backlog time

No backlog growth

Low CPU utilization

Downscaling Policy

Next lower scaling quanta => M' machines

Estimate future $CPU_{M'}$ per machine:

$$CPU_{M'} = \frac{M}{M'} CPU_M$$

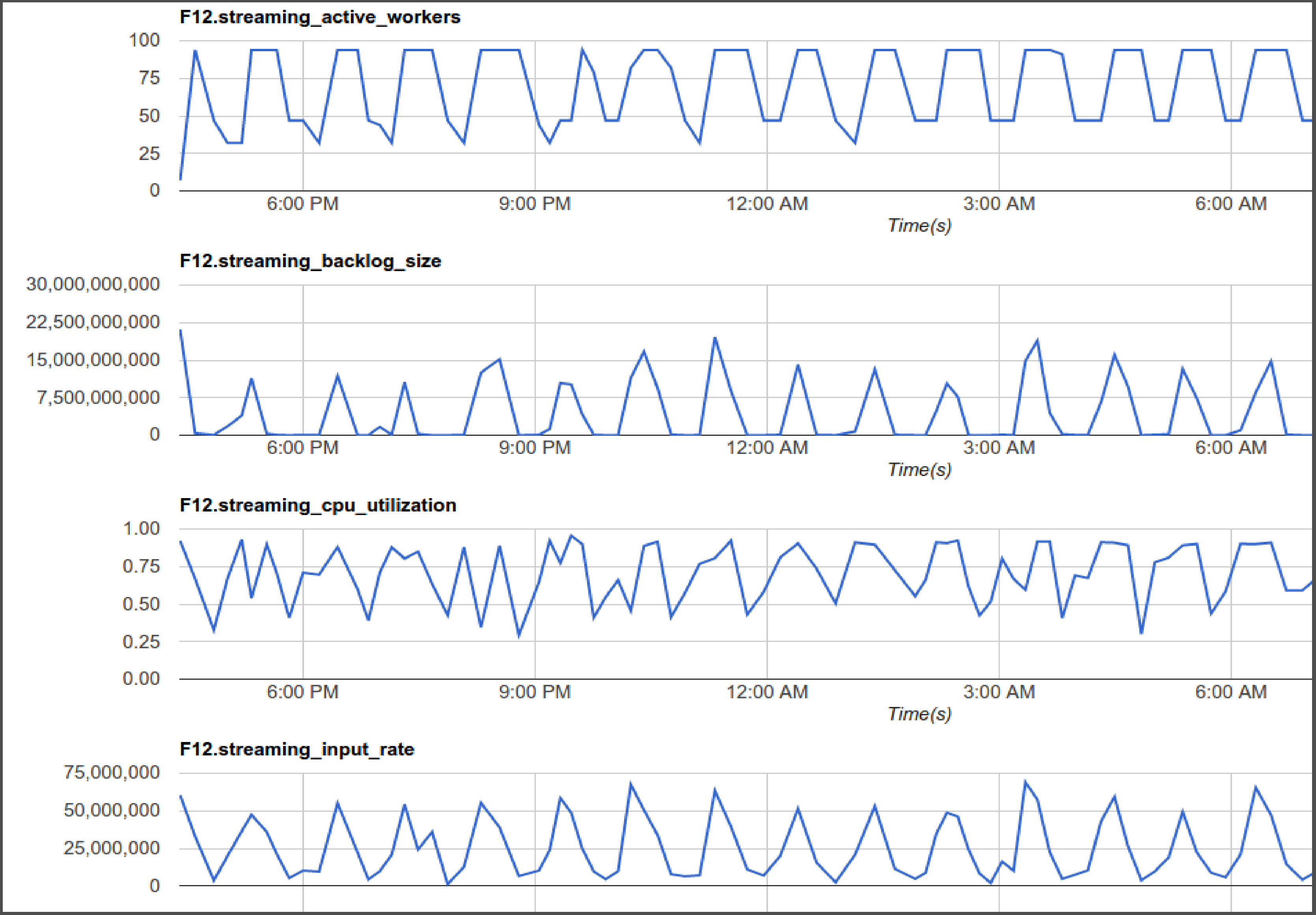
If new $CPU_{M'}$ < threshold (say 90%),

downscale to M'

4

Summary + Future Work

Artificial Experiment



Auto-Scaling Summary

Signals: throughput, backlog time, backlog growth, CPU utilization

Policy: keep up, reduce backlog, use CPUs

Mechanism: split key ranges, migrate computations

Future Work

- Experiment with non-uniform disk distributions to address hot ranges

- Dynamically splitting ranges finer than initially done.

- Approximate model of #VM - throughput relation

Questions?

Further reading on streaming model:

[The world beyond batch: Streaming 101]

[The world beyond batch: Streaming 102]