ORACLE®

ORACLE®

Towards Whatever-Scale Abstractions for Data-Driven Parallelism

Tim Harris, Maurice Herlihy, Yossi Lev, Yujie Liu,
Victor Luchangco, Virendra J. Marathe, Mark Moir

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Diversity

Blades have 100+ h/w threads, large machines have 1000s

T5-1B
16-cores
128GB-512GB DRAM

SuperCluster T5-8
2 * T5-8 compute nodes
QDR (40 Gb/sec) InfiniBand

SuperCluster M6-32
Up to 32 M6 processors
Up to 32 TB
Cache coherent interconnect

Diversity

Boundary becoming blurred between “machine” and “cluster”

Partial failures

Fast access times to data in RAM

Remote access to memory

Cache-coherent memory

Diversity

Heterogeneity between processor families

X64 (E5-2660)

8 cores
2 threads per core
256K L2 per core
20M shared L3
Turbo boost

SPARC (T5)

16 cores
8 strands per core
128K L2 per core
8M shared L3
2 out-of-order pipelines
1 FGU & Accelerators
Critical thread optimization

Specialized

…

ORACLE

Domino

An example whatever-scale abstraction

- Distributed shared memory model
- Data driven computation – tasks are triggered when data they watch is updated
- Phases – provide some control over when tasks are scheduled, avoid bad ordering
- Single address space implementation
- Control for asynchronous communication and waiting within a task
- NUMA & cluster implementation sketches

HotPar '13

WRSC '14

ORACLE

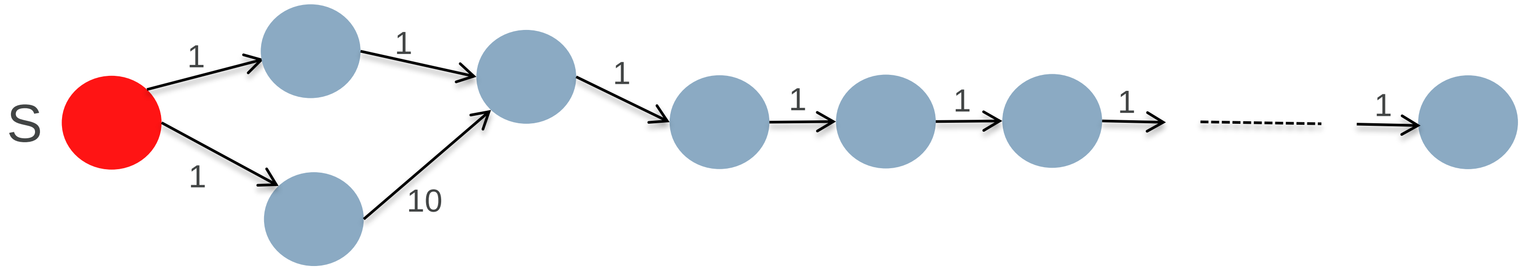
Domino

An example whatever-scale abstraction

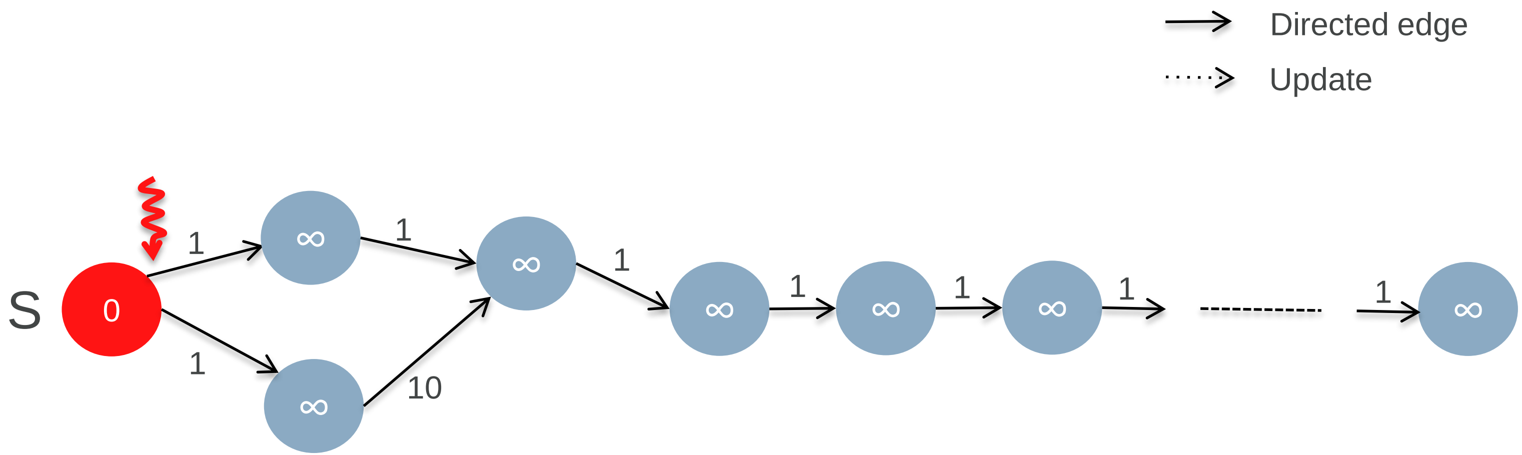
- Distributed shared memory model
- Data driven computation – tasks are triggered when data they watch is updated
- Phases – provide some control over when tasks are scheduled, avoid bad ordering
- Single address space implementation
- Control for asynchronous communication and waiting within a task
- NUMA & cluster implementation sketches

ORACLE

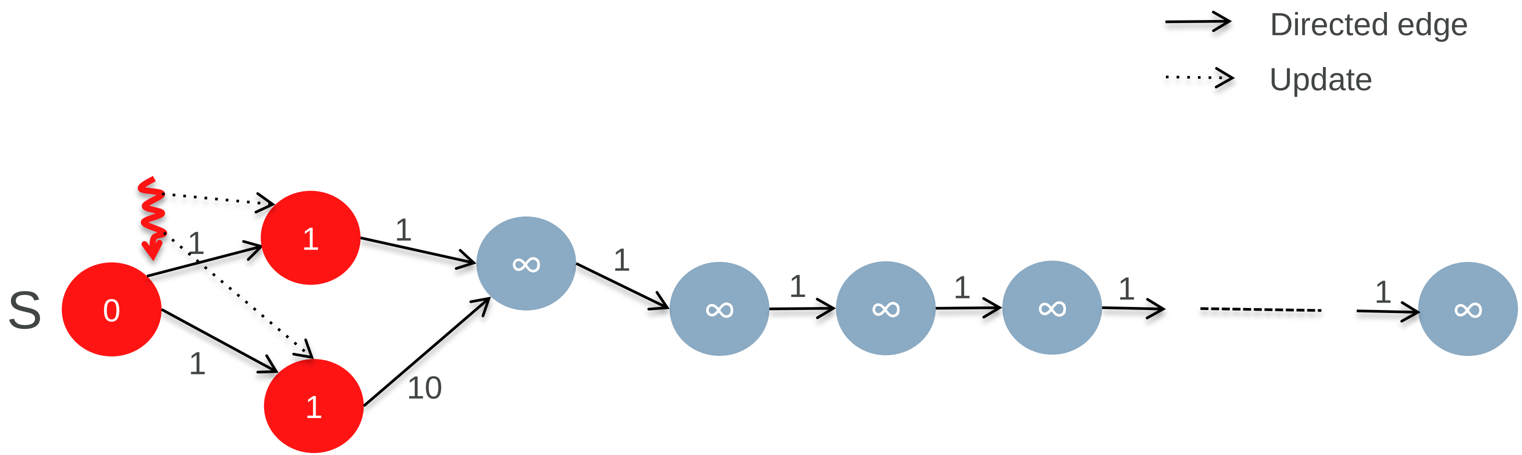
Recap: A Data-Driven SSSP algorithm



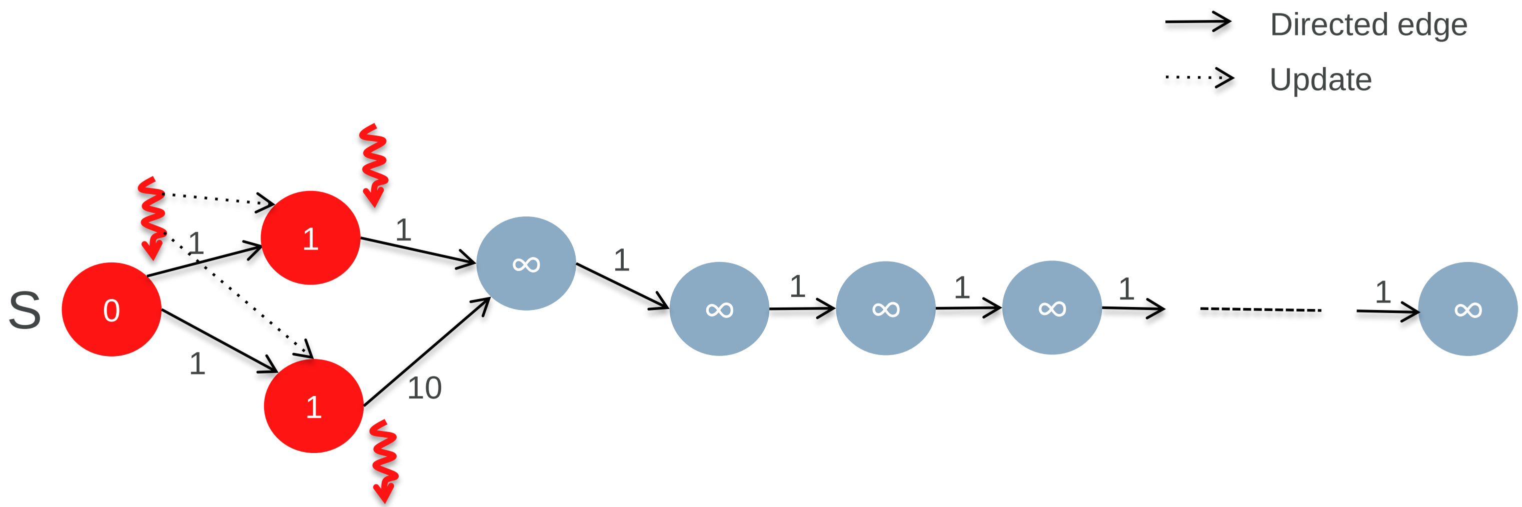
Recap: A Data-Driven SSSP algorithm



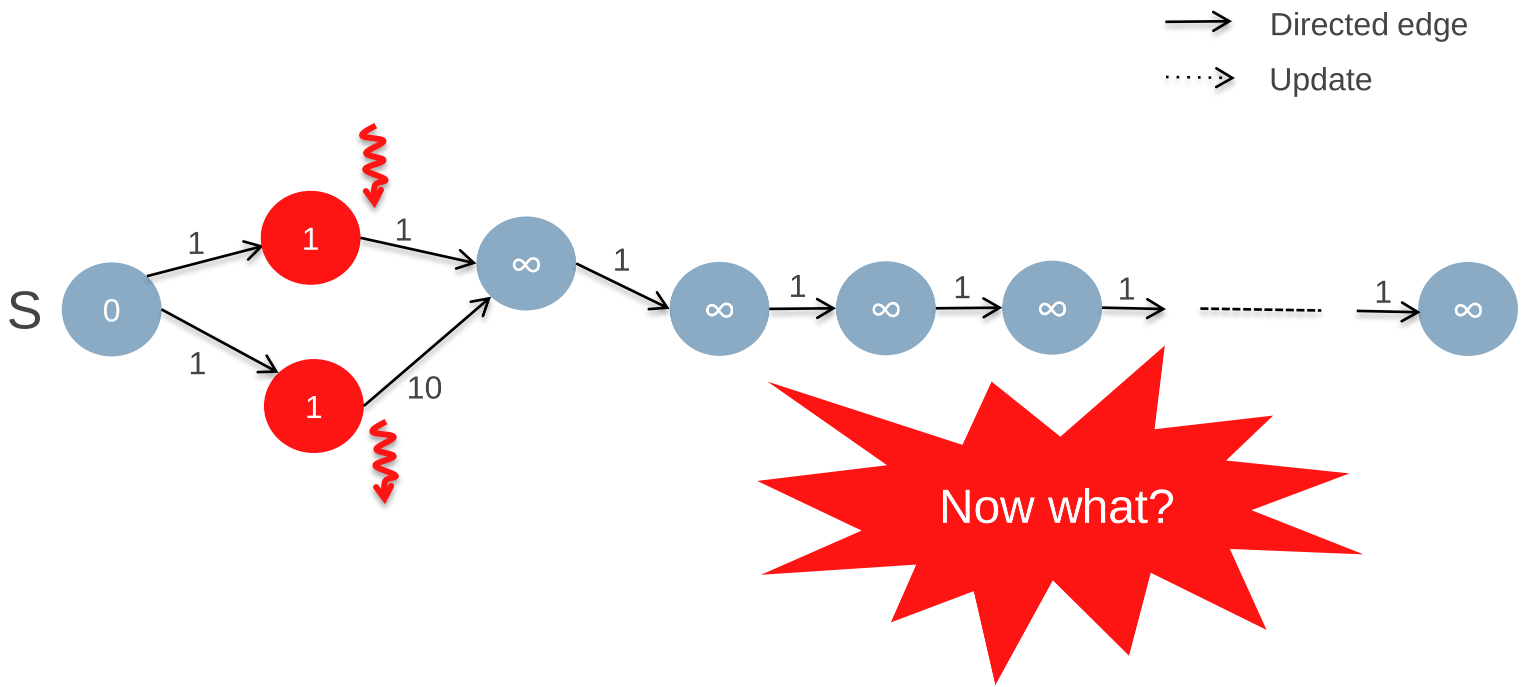
Recap: A Data-Driven SSSP algorithm



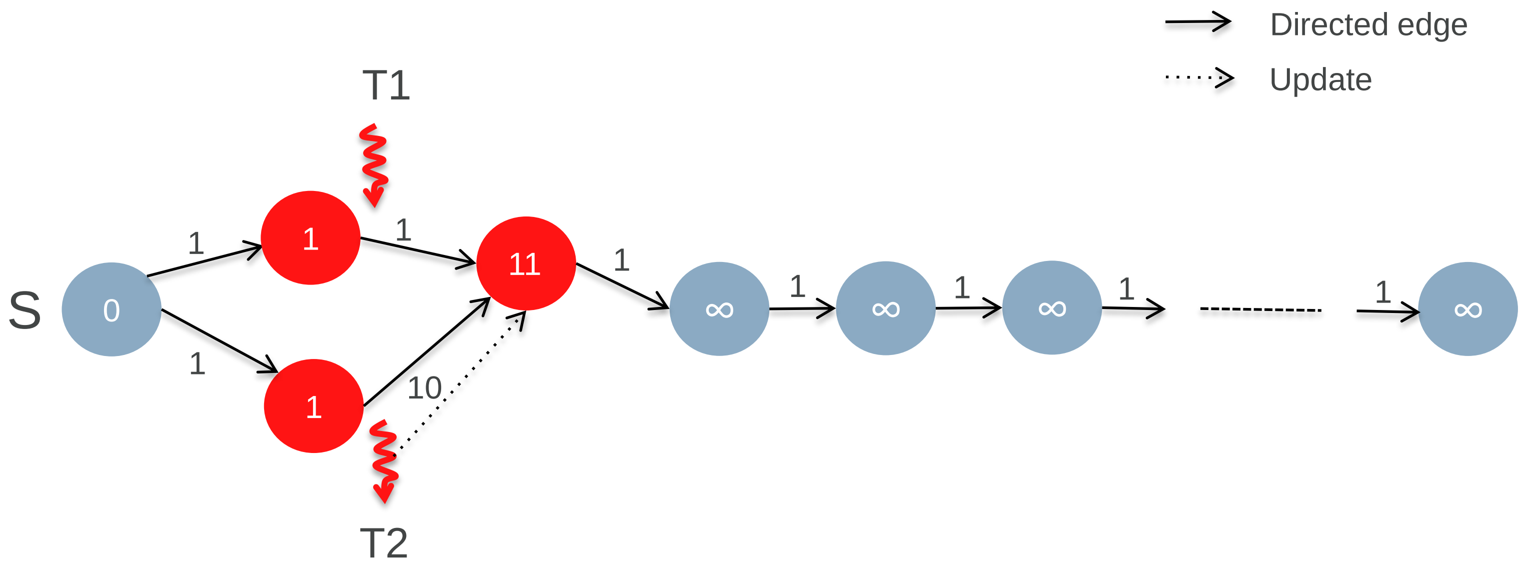
Recap: A Data-Driven SSSP algorithm



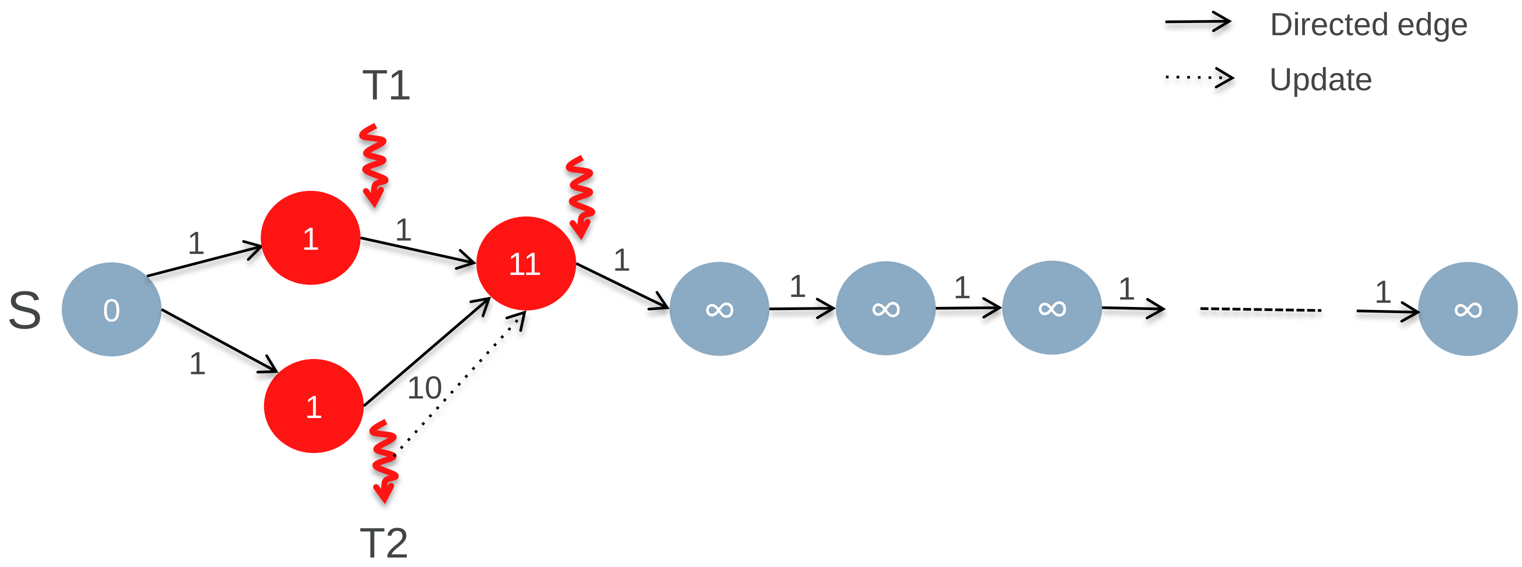
Recap: A Data-Driven SSSP algorithm



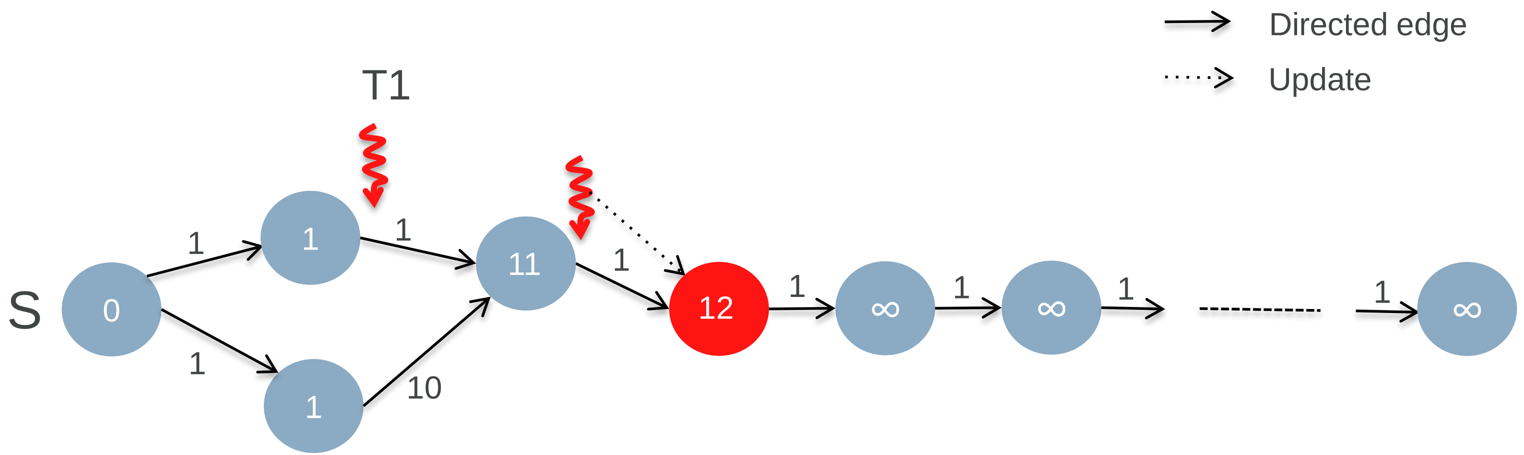
Recap: A Data-Driven SSSP algorithm



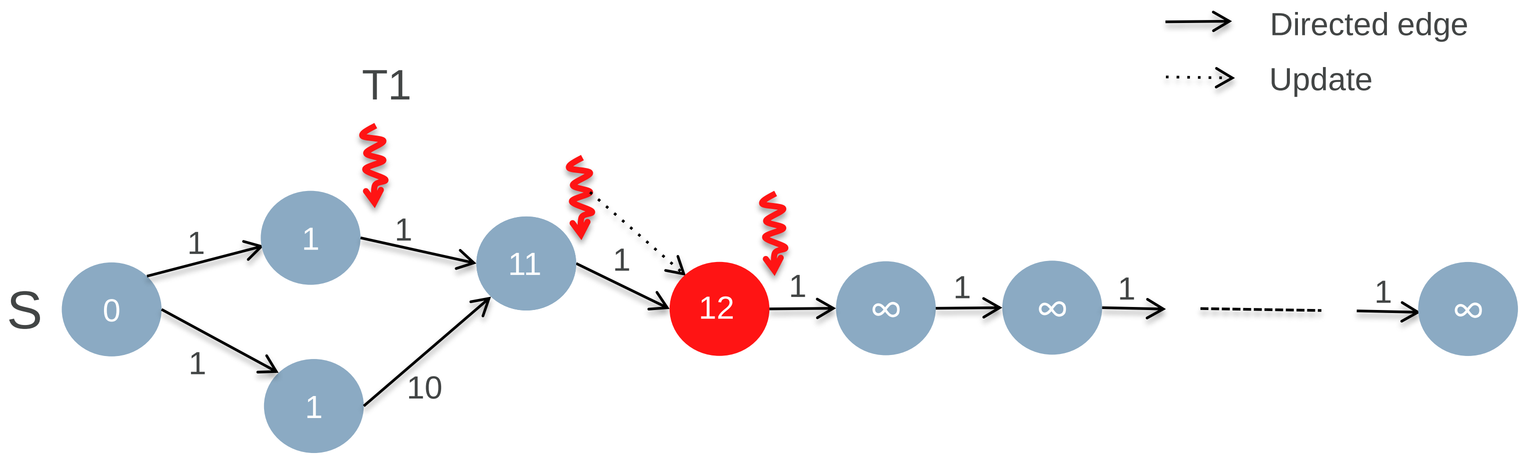
Recap: A Data-Driven SSSP algorithm



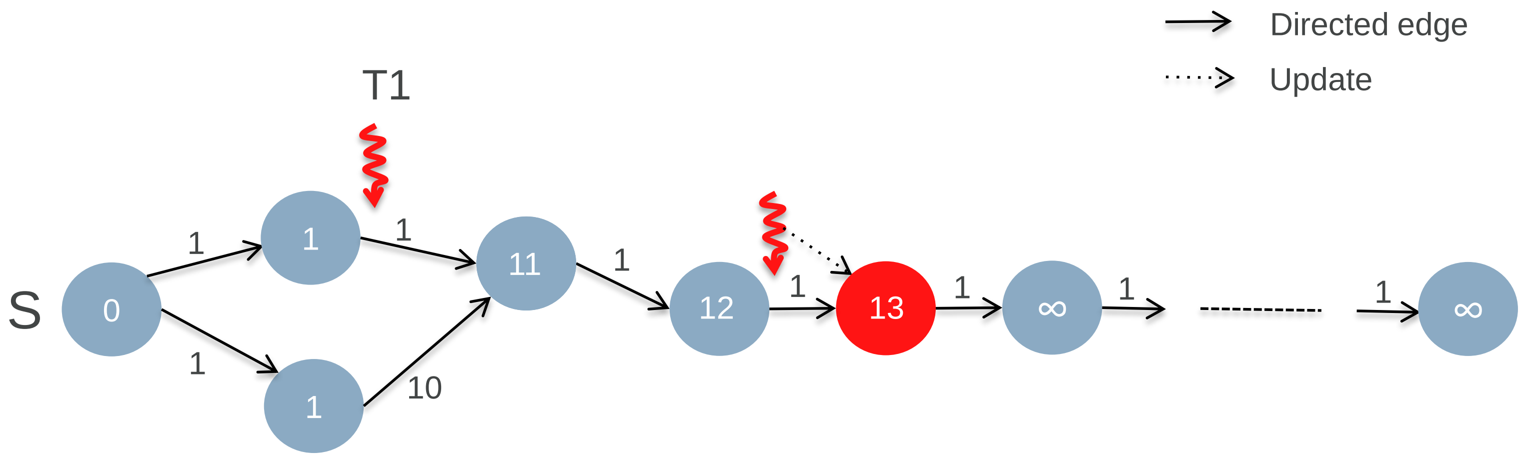
Recap: A Data-Driven SSSP algorithm



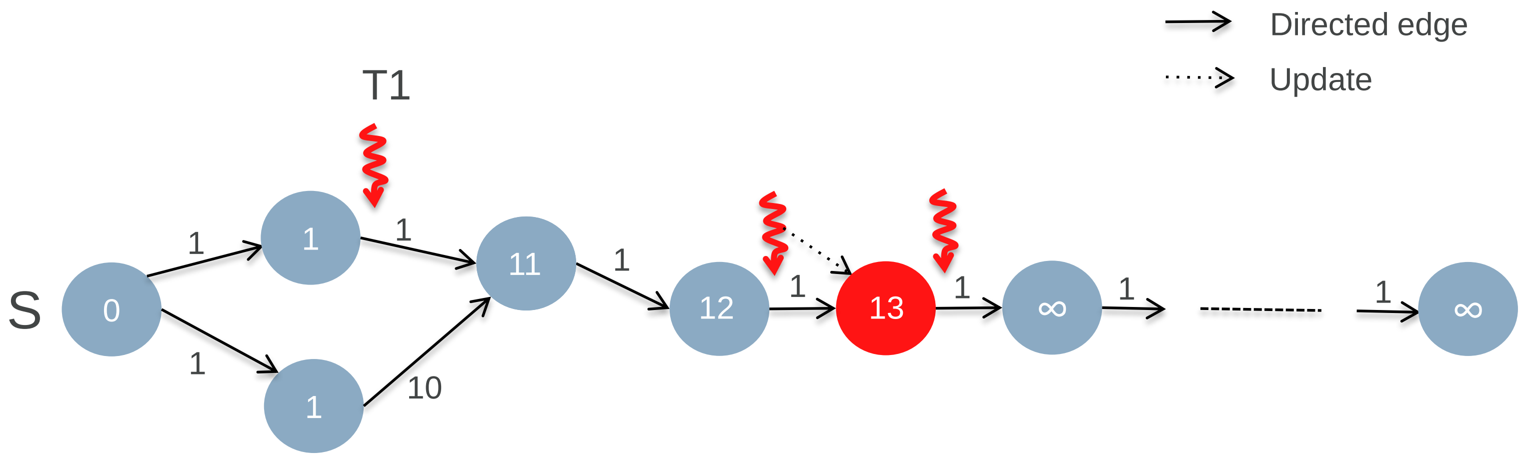
Recap: A Data-Driven SSSP algorithm



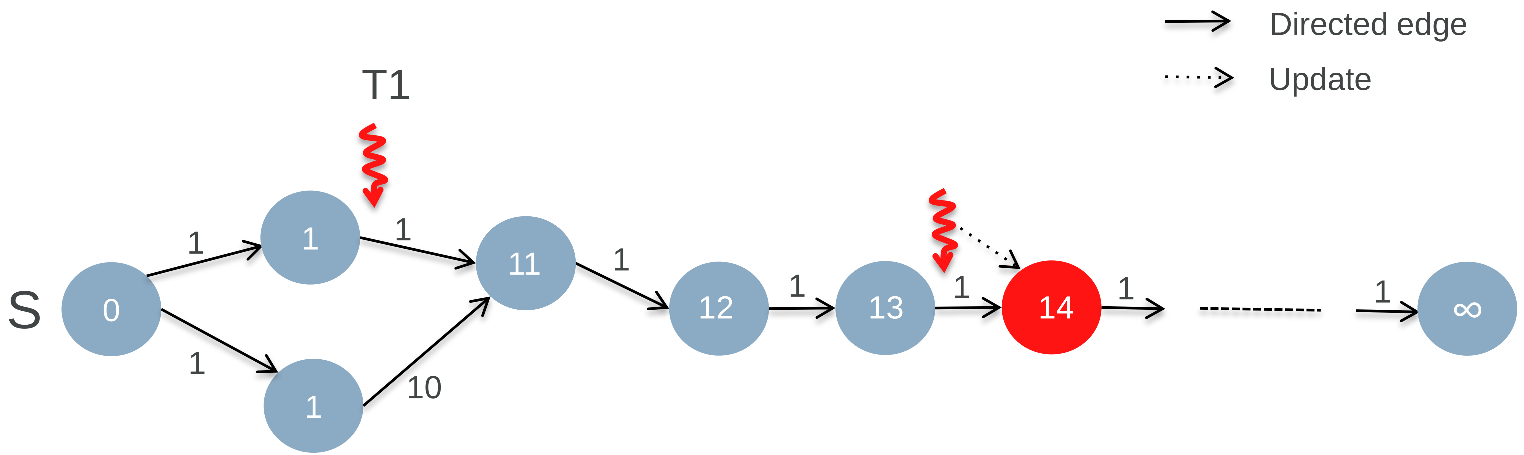
Recap: A Data-Driven SSSP algorithm



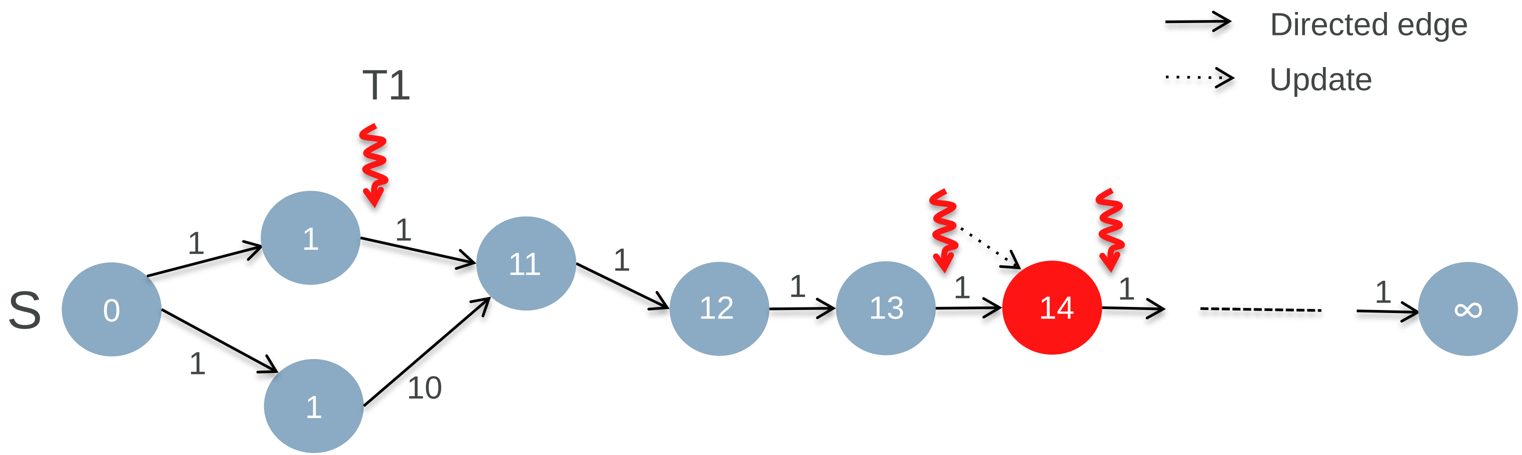
Recap: A Data-Driven SSSP algorithm



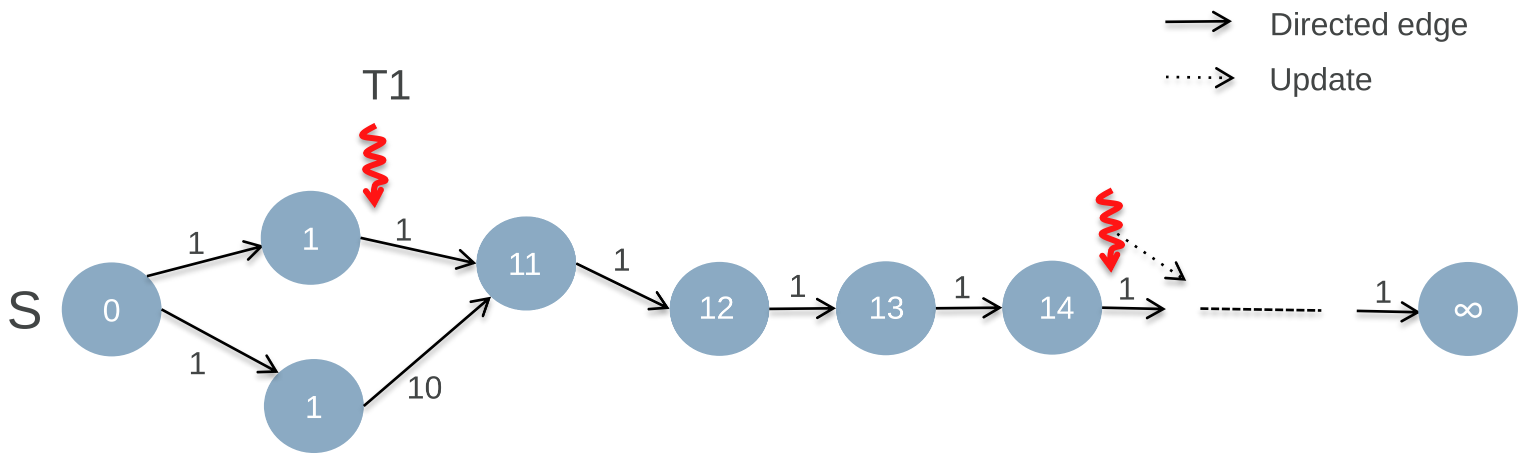
Recap: A Data-Driven SSSP algorithm



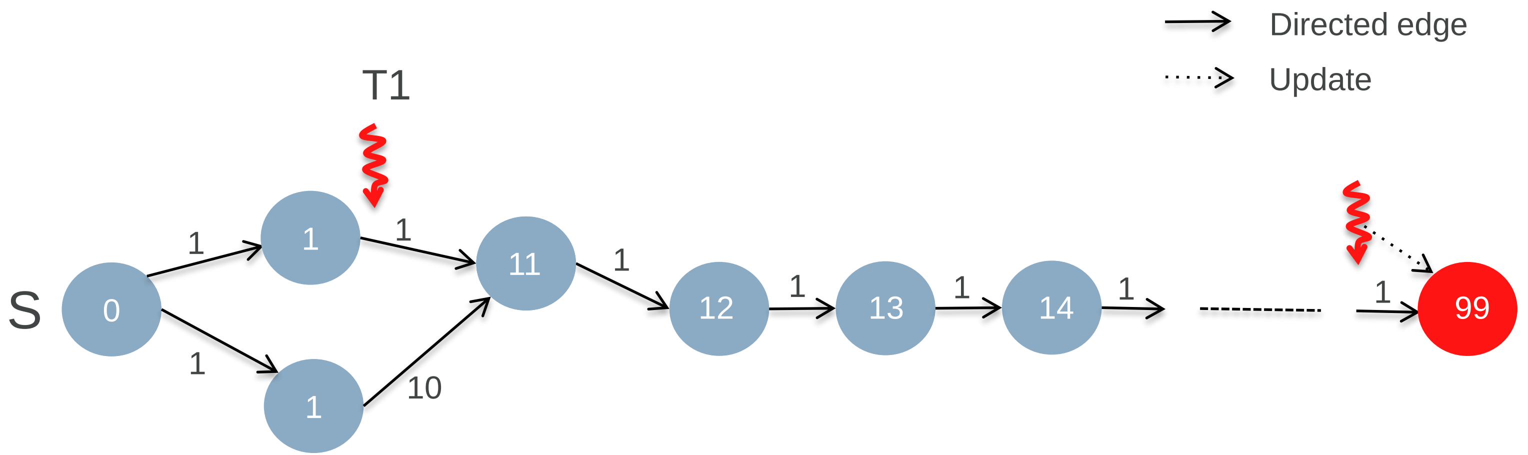
Recap: A Data-Driven SSSP algorithm



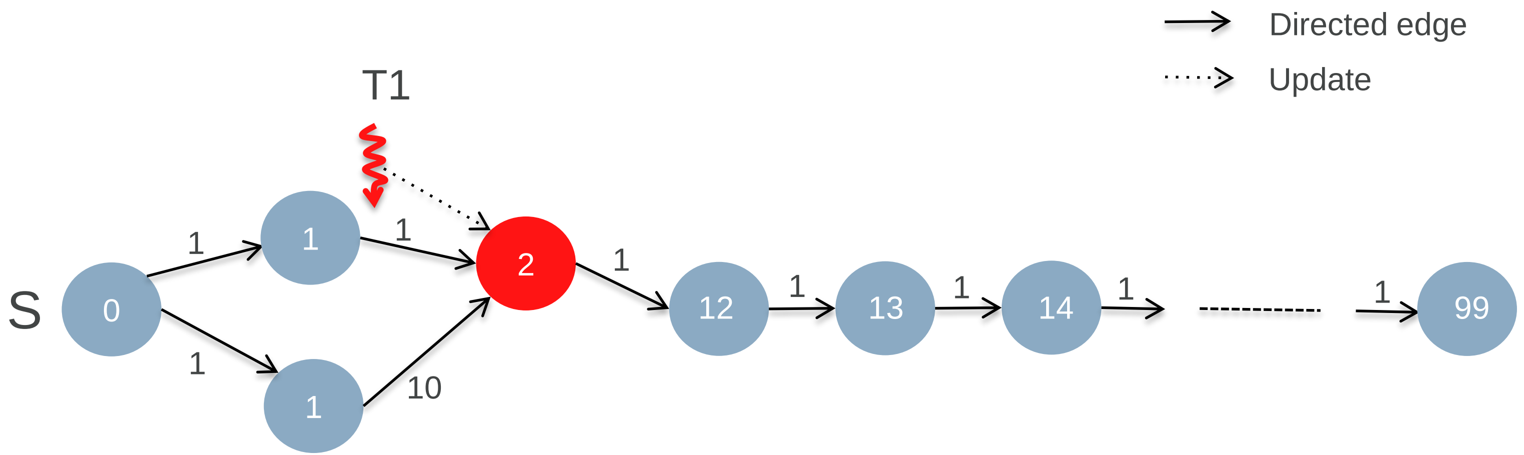
Recap: A Data-Driven SSSP algorithm



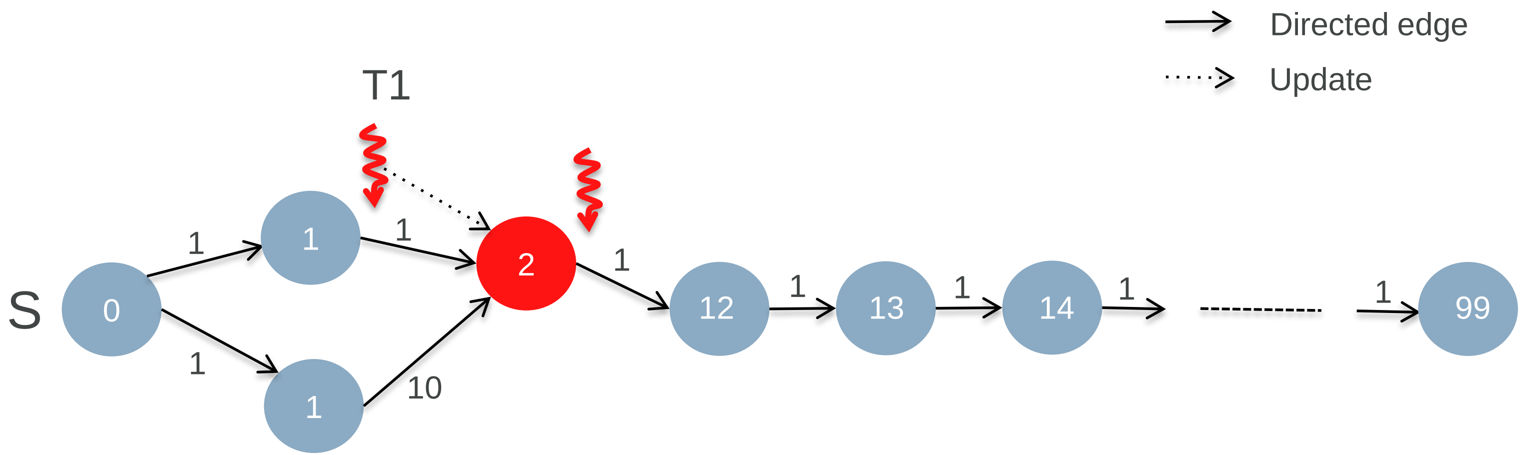
Recap: A Data-Driven SSSP algorithm



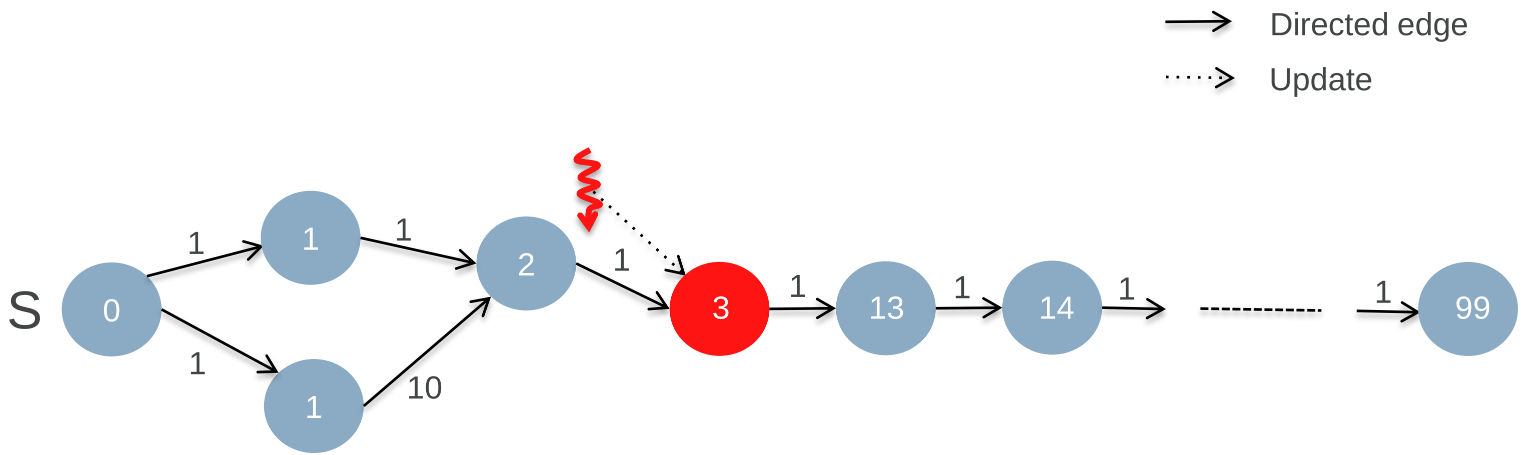
Recap: A Data-Driven SSSP algorithm



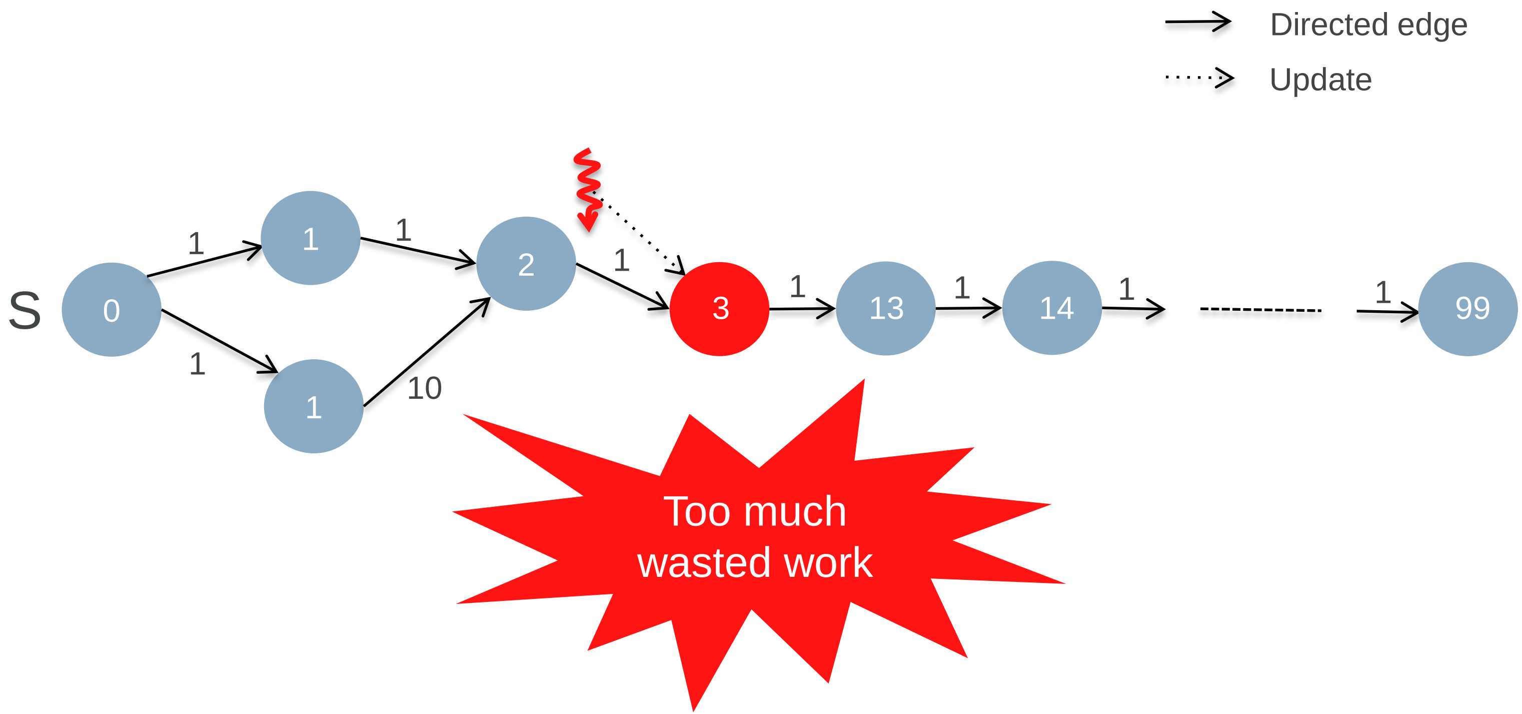
Recap: A Data-Driven SSSP algorithm



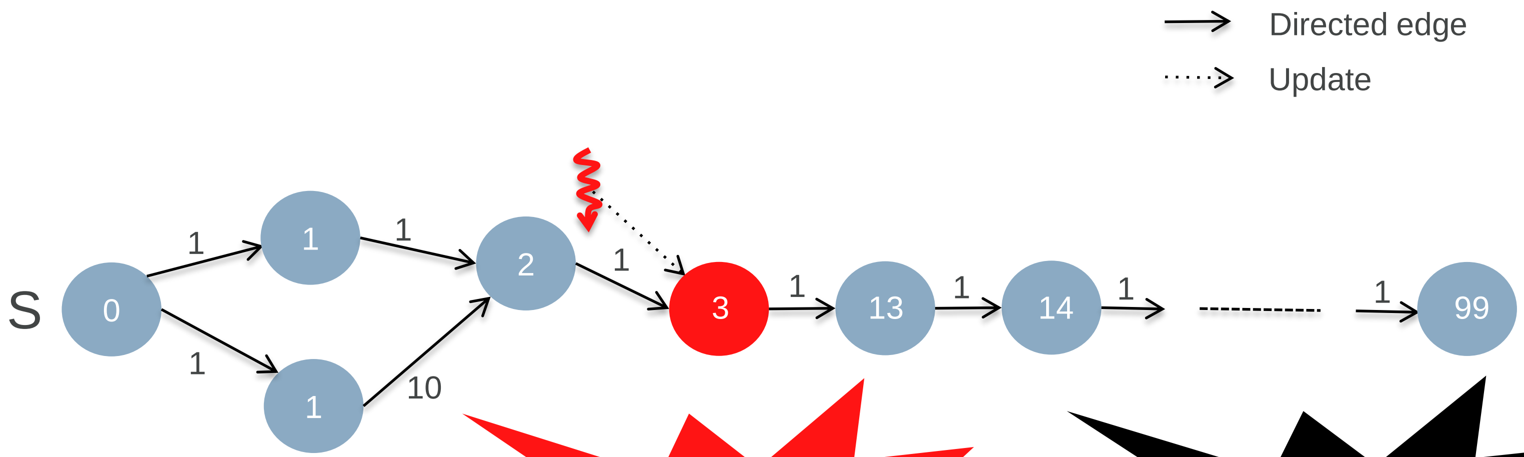
Recap: A Data-Driven SSSP algorithm

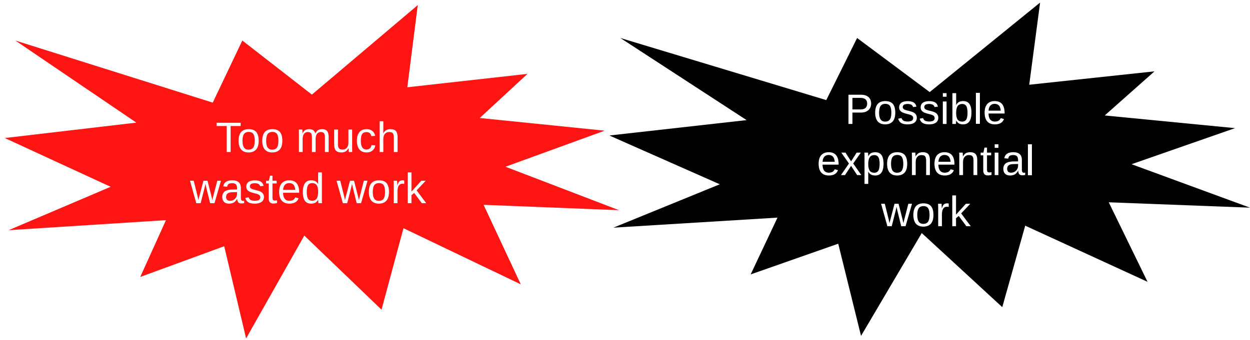


Recap: A Data-Driven SSSP algorithm

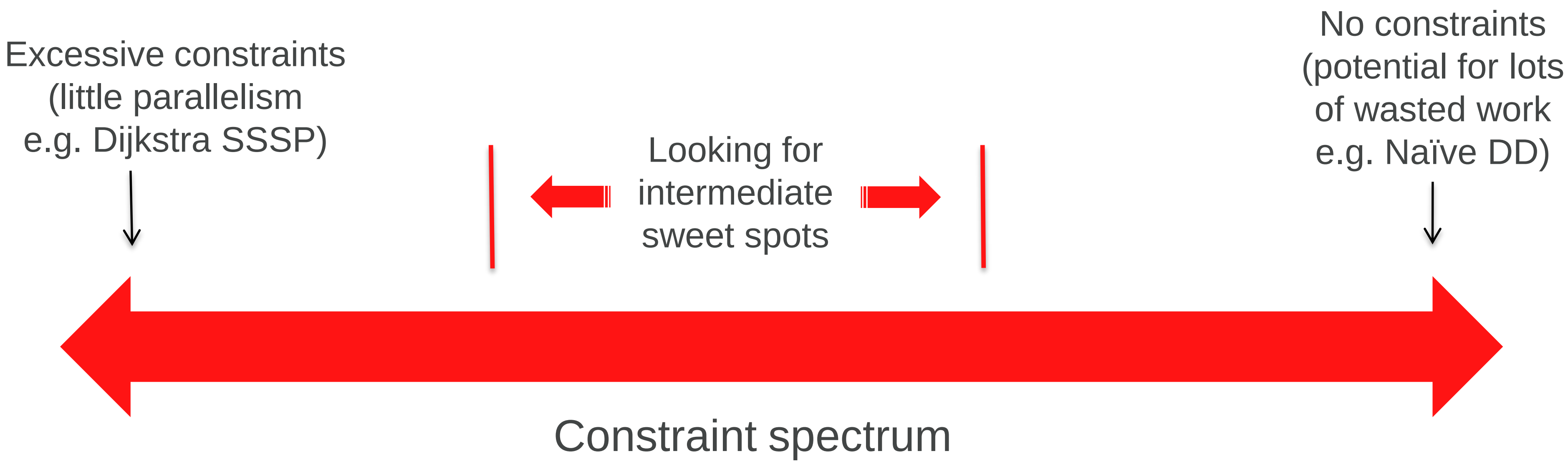


Recap: A Data-Driven SSSP algorithm





Recap: A Data-Driven SSSP algorithm



Recap: A Data-Driven SSSP algorithm

"deferred" triggers and phases

- Pseudo-code:

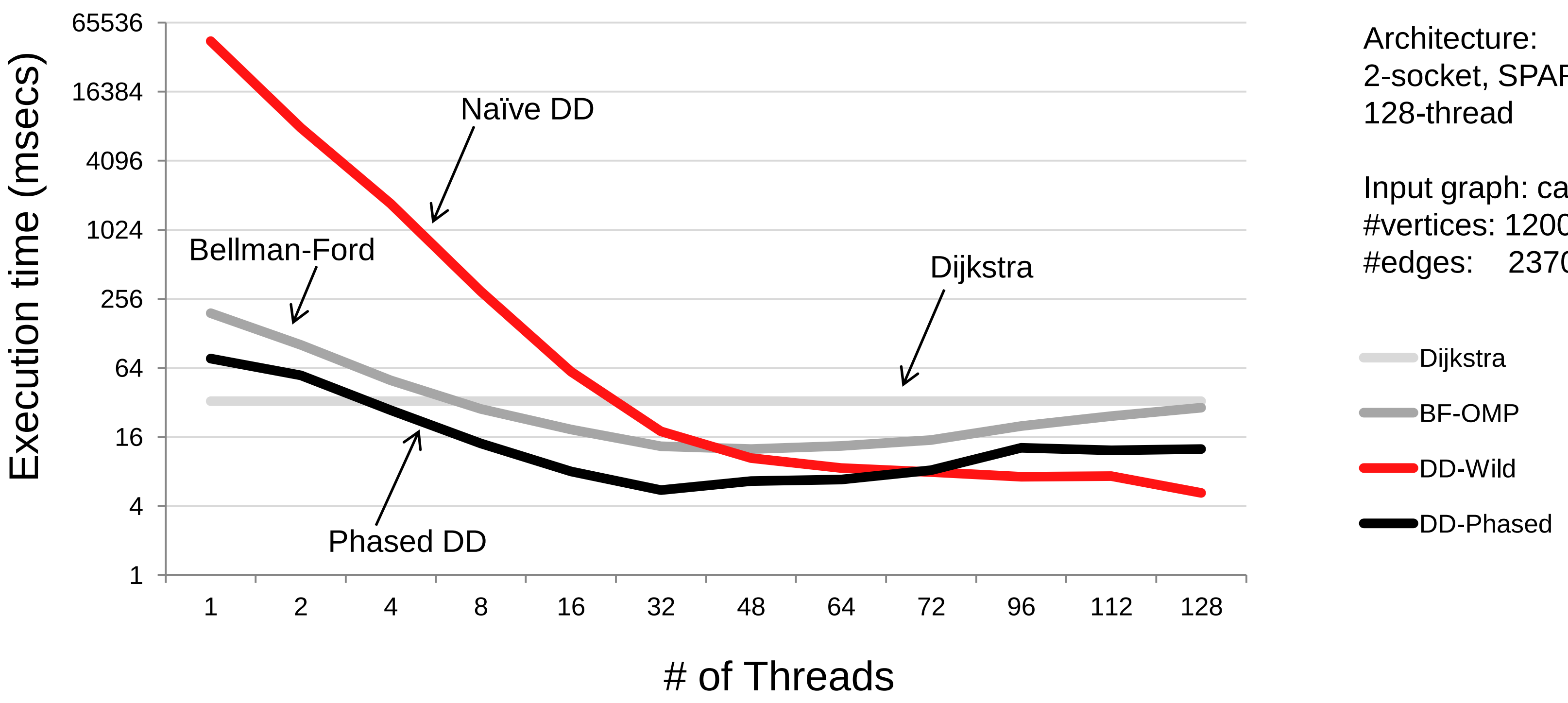
  x triggers f() --- writing to x creates a task to run this.f() in the current phase

  x triggers deferred f() – writing to x creates a task to run this.f() in the next phase

- Semantics: single sequence of phases, no task in phase N+1 starts until phase N is complete

ORACLE

Recap: A Data-Driven SSSP algorithm



Architecture:
2-socket, SPARC T2+
128-thread

Input graph: ca-HepPh
#vertices: 12008
#edges: 237042

Domino

An example whatever-scale abstraction

- Distributed shared memory model
- Data driven computation – tasks are triggered when data they watch is updated
- Phases – provide some control over when tasks are scheduled, avoid bad cases
- Single address space implementation
- Control for asynchronous communication and waiting within a task
- NUMA & cluster implementation sketches

Programming model

Distributed objects with synchronous RPC

```
class Node {
  int v triggers deferred compute();


}
```

SSSP example – each graph node holds its current distance "v" from the root, and updates to the distance trigger the method "compute" to be run in the next phase.

Programming model

Distributed objects with synchronous RPC

```
class Node {
  int v triggers deferred compute();

  gRef Node[] neighbors;
```

“neighbors” is an array of “global-ref-to-Node”, identifying possibly-remote objects

```
}
```

Programming model

Distributed objects with synchronous RPC

```
class Node {
  int v triggers deferred compute();

  gRef Node[] neighbors;

  void compute() {
    for (int i = 0; i < numNeighbors; i ++) {
      neighbors[i].updateDistance(v+1);
    }
  }
}
```

Simple synchronous implementation, calling an “updateDistance” method on each neighbor (which in turn may write to “v” on that object)

ORACLE

Programming model

“async” and “do…finish”

```
…

void compute() {

    for (int i = 0; i < numNeighbors; i ++) {
       neighbors[i].updateDistance(v+1);
    }

}

…
```

Programming model

“async” and “do…finish”

```
…

void compute() {
  do {
    for (int i = 0; i < numNeighbors; i ++) {
      async neighbors[i].updateDistance(v+1);
    }
  } finish;
}

…
```

async: if any of these calls needs to wait for RPC, then execution can proceed through the rest of the loop

ORACLE

Programming model

“async” and “do…finish”

```
…

void compute() {
  do {
    for (int i = 0; i < numNeighbors; i ++) {
      async neighbors[i].updateDistance(v+1);
    }
  } finish;
}

…
```

async: if any of these calls needs to wait for RPC, then execution can proceed through the rest of the loop

finish: execution cannot proceed past here until all of the RPCs complete

Programming model

Design decisions

- As in Barrelfish:
  - “async” work is independent of threading
  - “async” must be statically within “do/finish”
  - Only switch on blocking
- We do not need concurrency control on local variables
- Locals captured by “async” will remain alive
  - A simple cactus-stack implementation is sufficient
- In the absence of blocking, “synchronous elision” only valid behavior

ORACLE

Three different scale implementations

Single-machine SMP

- Run within a single address space
- “gRef T” is just a “*T”
- “RPC” is just a normal method call
- “do/finish” and “async” are ignored
- Pool of worker threads with per-worker dequeues
- Work-stealing for load balancing
- SNZI objects used to track work in phases

ORACLE

Three different scale implementations

Single-machine NUMA

- Run within a single address space
- Logically distribute objects between NUMA domains
- A “gRef T” holds a NUMA domain ID and a bare pointer
- Cross-domain operations on gRefs use message passing
- Currently, “do/finish”, and “async” built manually using call-backs and a “split task” abstraction
- Separate worker threads in each NUMA domain
- Separate SNZI objects in each NUMA domain, plus a shared top-level counter

Three different scale implementations

Cluster with InfiniBand

- Retain same structure as NUMA, except:
- Use RDMA to transfer batches of RPC requests/responses
- Cannot rely on shared top-level counter for detecting phase changes

ORACLE

Concluding thoughts

Implementation and practical evaluation is work-in-progress

- Work in progress
- To what extent do we need async/do-finish in the distributed case?
  - Two sources of parallelism
  - Do we need both?
- Should we relax the “phase” concept?
  - Allow two adjacent phases to run concurrently?
  - How much synchronization is needed to avoid poor performance?
  - Can we combine this synchronization with messages needed for RPC?

ORACLE

ORACLE®