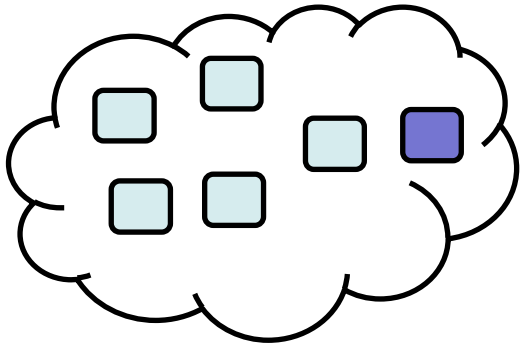
Update Propagation in a Streaming Warehouse

Theodore Johnson
Vladislav Shkapenyuk

SSDBM 2011

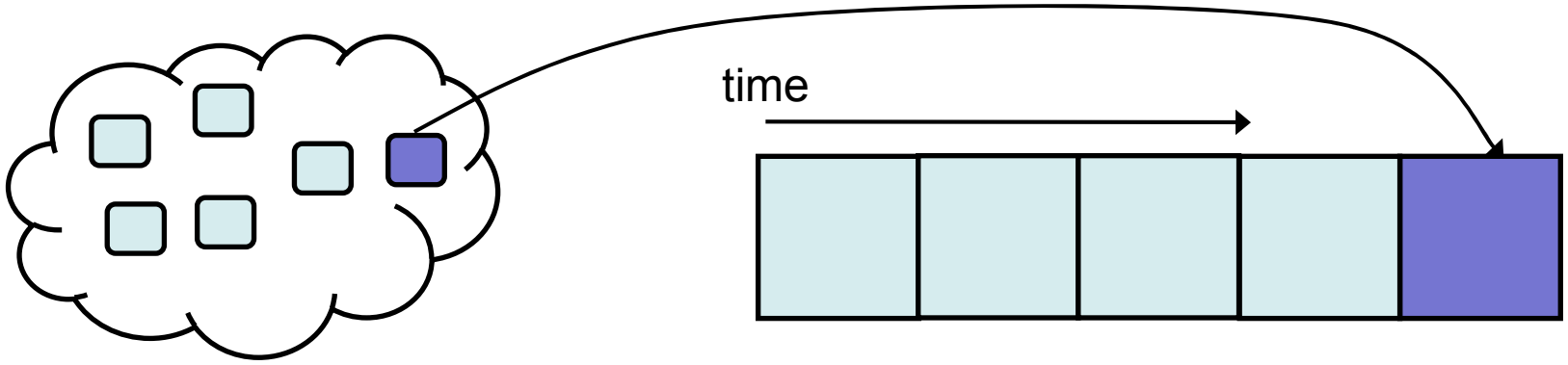
Managing a Stream Warehouse



Raw data

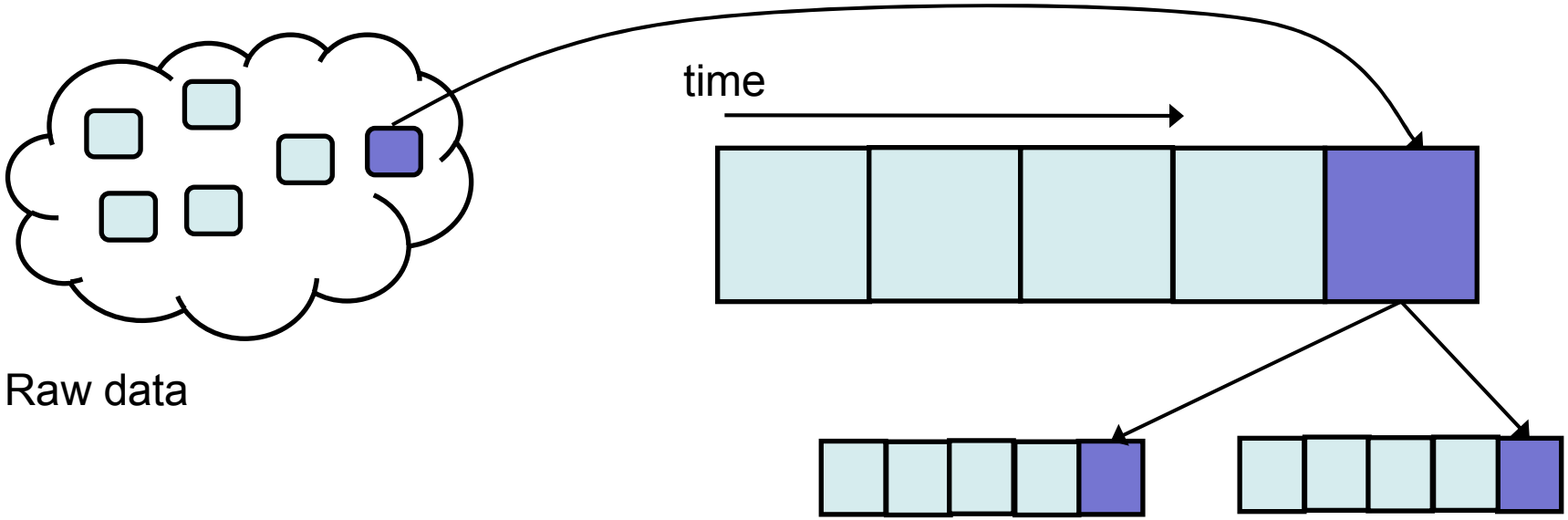
- Continually arriving data

Managing a Stream Warehouse



- Continually arriving data
- Is loaded into temporally partitioned base tables

Managing a Stream Warehouse

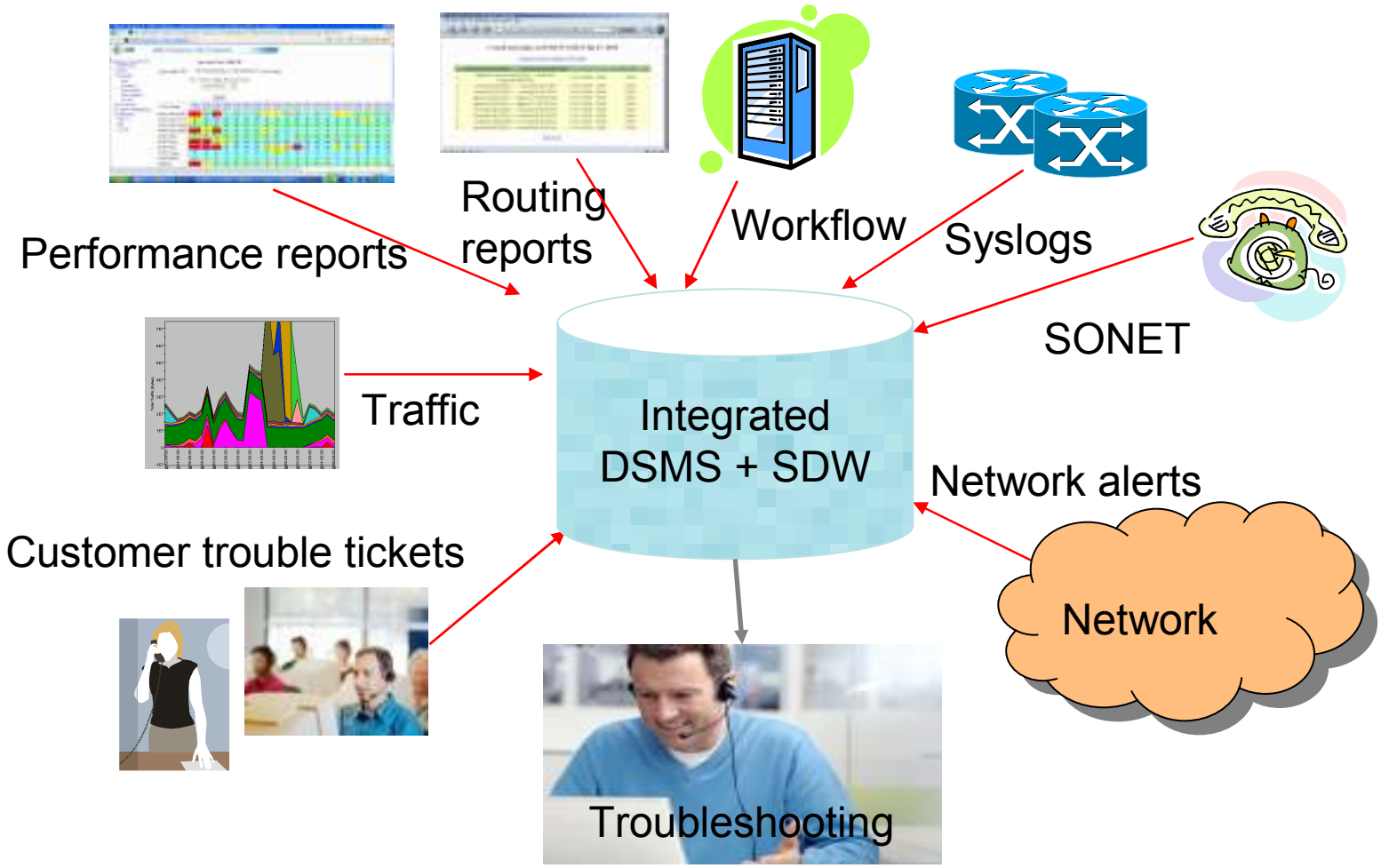


- Continually arriving data
- Is loaded into temporally partitioned base tables
- Updates propagate to higher level data products.

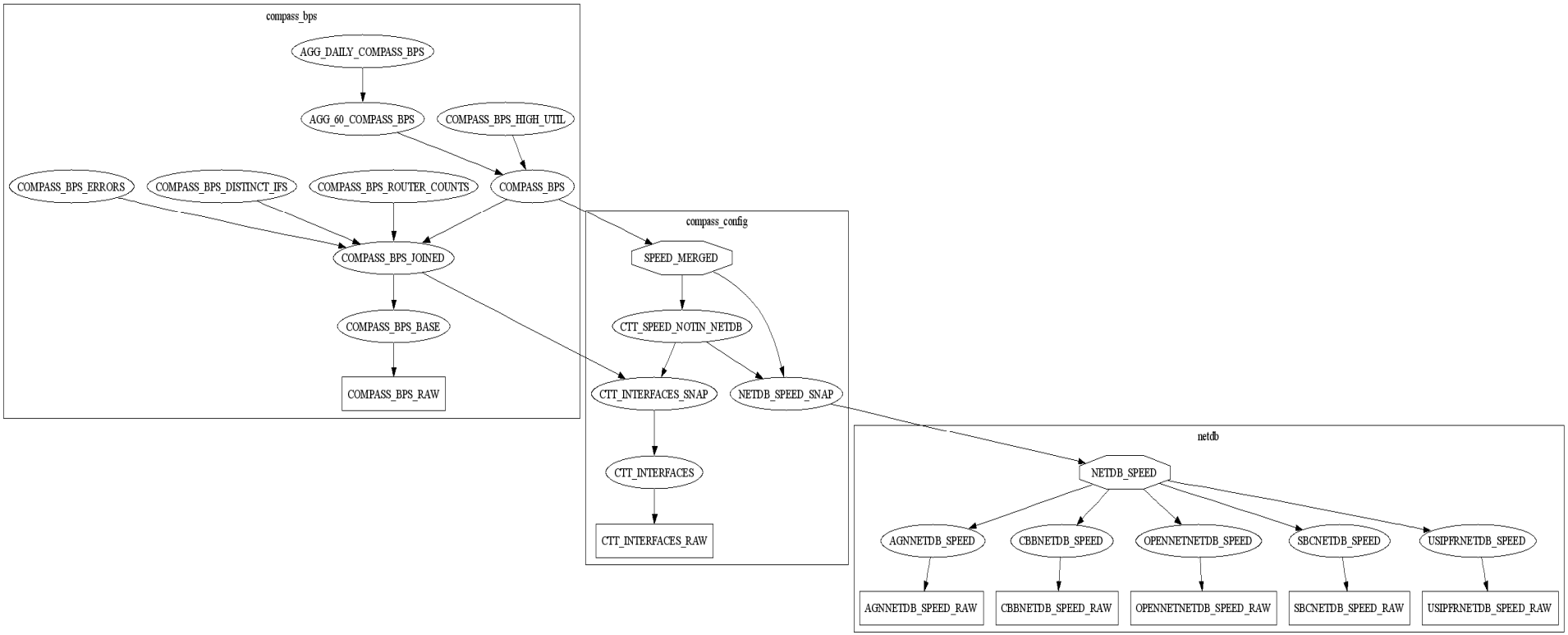
DataDepot

- DataDepot is a stream warehouse generator
  - On top of the Daytona database.
- Applications within AT&T
  - Darkstar warehouse: Network performance, configuration, and alert data.
  - Other petabyte-sized applications.
- Materialized view maintenance via update propagation through partitions.
- Near-real time: Propagate updates as soon as possible
  - Don’t wait for global updates to materialized views.
- Multiversion Concurrency Control

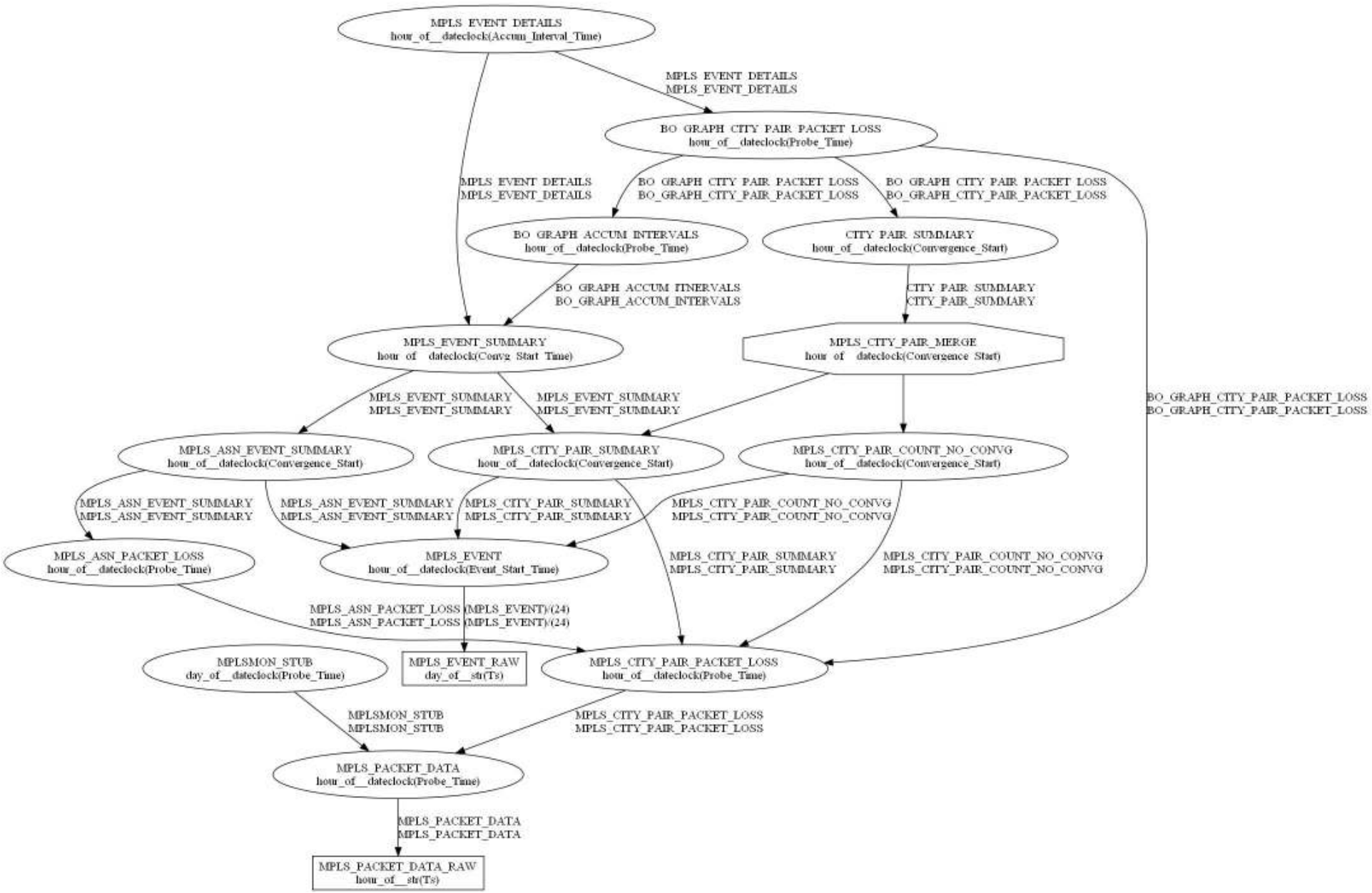
Darkstar: Backbone Network Data.



BPS


compass_bps
AGG_DAILY_COMPASS_BPS
AGG_60_COMPASS_BPS
COMPASS_BPS_HIGH_UTIL
COMPASS_BPS_ERRORS
COMPASS_BPS_DISTINCT_IFS
COMPASS_BPS_ROUTER_COUNTS
COMPASS_BPS
COMPASS_BPS_JOINED
COMPASS_BPS_BASE
COMPASS_BPS_RAW
compass_config
SPEED_MERGED
CTT_SPEED_NOTIN_NETDB
CTT_INTERFACES_SNAP
NETDB_SPEED_SNAP
CTT_INTERFACES
CTT_INTERFACES_RAW
netdb
NETDB_SPEED
AGNNETDB_SPEED
CBBNETDB_SPEED
OPENNETNETDB_SPEED
SBCNETDB_SPEED
USIPFRNETDB_SPEED
AGNNETDB_SPEED_RAW
CBBNETDB_SPEED_RAW
OPENNETNETDB_SPEED_RAW
SBCNETDB_SPEED_RAW
USIPFRNETDB_SPEED_RAW

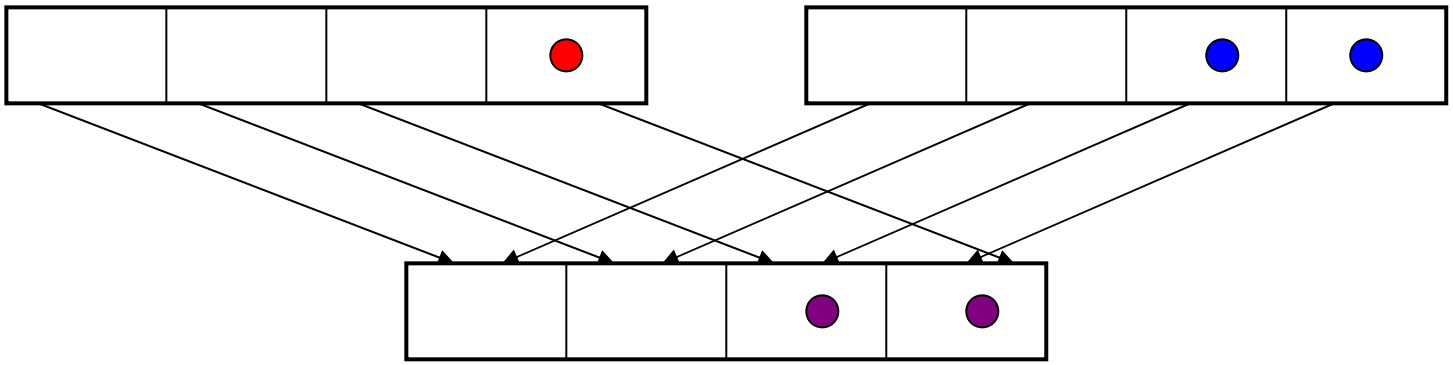
MPLS_MON


MPLS EVENT DETAILS
hour_of__dateclock(Accum_Interval_Time)
MPLS EVENT DETAILS
MPLS_EVENT_DETAILS
BO GRAPH CITY PAIR PACKET LOSS
hour_of__dateclock(Probe_Time)
MPLS EVENT DETAILS
MPLS_EVENT_DETAILS
BO GRAPH CITY PAIR PACKET LOSS
BO_GRAPH_CITY_PAIR_PACKET_LOSS
BO GRAPH CITY PAIR PACKET LOSS
BO_GRAPH_CITY_PAIR_PACKET_LOSS
BO GRAPH ACCUM INTERVALS
hour_of__dateclock(Probe_Time)
CITY PAIR SUMMARY
hour_of__dateclock(Convergence_Start)
BO GRAPH ACCUM INTERVALS
BO_GRAPH_ACCUM_INTERVALS
CITY PAIR SUMMARY
CITY_PAIR_SUMMARY
MPLS_EVENT_SUMMARY
hour_of__dateclock(Convg_Start_Time)
MPLS_CITY_PAIR_MERGE
hour_of__dateclock(Convergence_Start)
MPLS_EVENT_SUMMARY
MPLS_EVENT_SUMMARY
MPLS_EVENT_SUMMARY
MPLS_EVENT_SUMMARY
BO_GRAPH_CITY_PAIR_PACKET_LOSS
BO_GRAPH_CITY_PAIR_PACKET_LOSS
MPLS_ASN_EVENT_SUMMARY
hour_of__dateclock(Convergence_Start)
MPLS_CITY_PAIR_SUMMARY
hour_of__dateclock(Convergence_Start)
MPLS_CITY_PAIR_COUNT_NO_CONVG
hour_of__dateclock(Convergence_Start)
MPLS_ASN_EVENT_SUMMARY
MPLS_ASN_EVENT_SUMMARY
MPLS_ASN_EVENT_SUMMARY
MPLS_ASN_EVENT_SUMMARY
MPLS_CITY_PAIR_SUMMARY
MPLS_CITY_PAIR_SUMMARY
MPLS_CITY_PAIR_COUNT_NO_CONVG
MPLS_CITY_PAIR_COUNT_NO_CONVG
MPLS_ASN_PACKET_LOSS
hour_of__dateclock(Probe_Time)
MPLS_EVENT
hour_of__dateclock(Event_Start_Time)
MPLS_CITY_PAIR_SUMMARY
MPLS_CITY_PAIR_SUMMARY
MPLS_CITY_PAIR_COUNT_NO_CONVG
MPLS_CITY_PAIR_COUNT_NO_CONVG
MPLS_ASN_PACKET_LOSS (MPLS_EVENT)/(24)
MPLS_ASN_PACKET_LOSS (MPLS_EVENT)/(24)
MPLSMON_STUB
day_of__dateclock(Probe_Time)
MPLS_EVENT_RAW
day_of__str(Ts)
MPLS_CITY_PAIR_PACKET_LOSS
hour_of__dateclock(Probe_Time)
MPLSMON_STUB
MPLSMON_STUB
MPLS_CITY_PAIR_PACKET_LOSS
MPLS_CITY_PAIR_PACKET_LOSS
MPLS_PACKET_DATA
hour_of__dateclock(Probe_Time)
MPLS_PACKET_DATA
MPLS_PACKET_DATA
MPLS_PACKET_DATA_RAW
hour_of__str(Ts)

Update Propagation

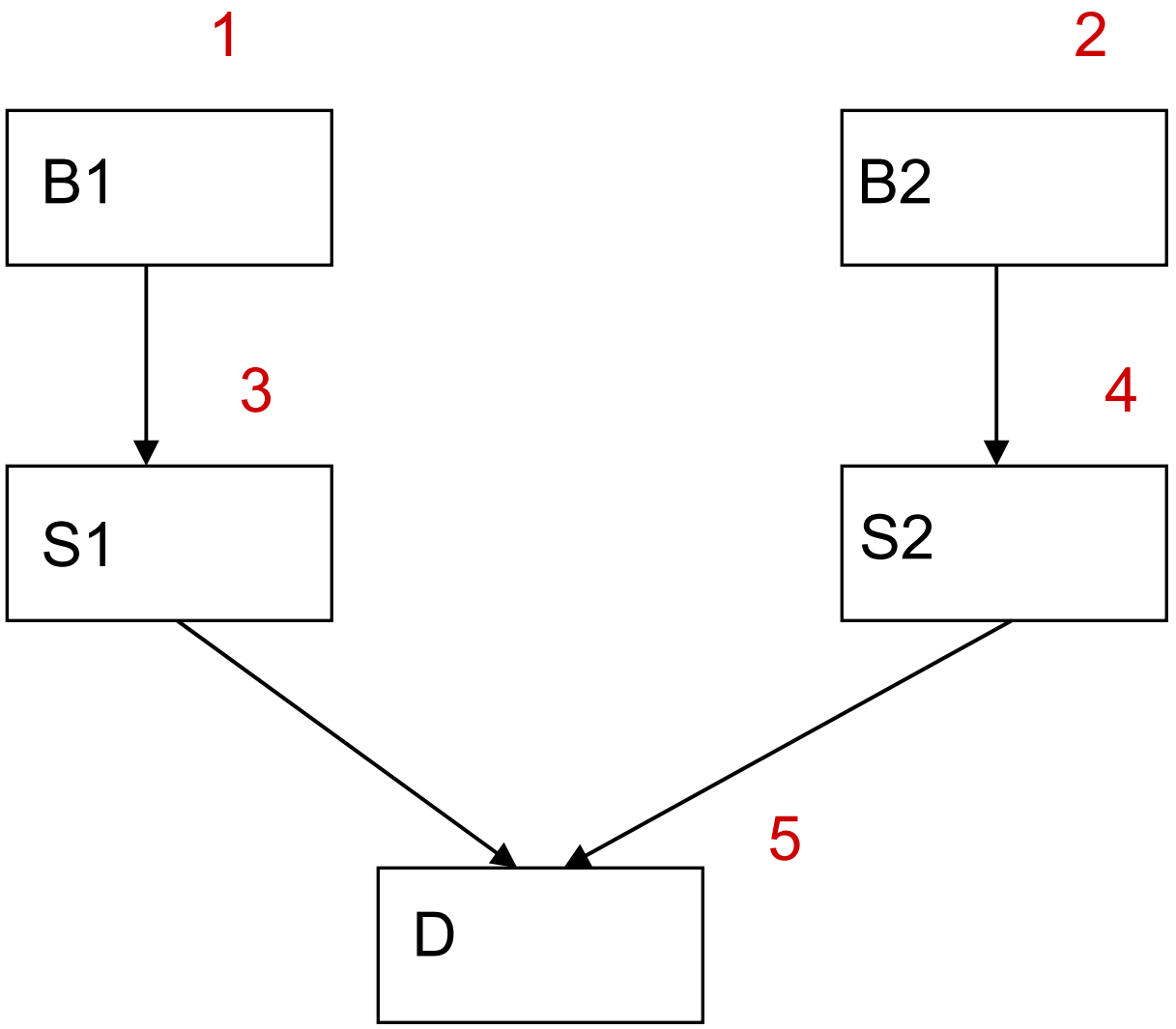
- We can build complex apps if we're confident that *all* updates get propagated.
- 1st version: used *make*-style algorithm
  - Not correct for complex configurations
  - Requires global analysis and all-at-once updates.
- Developed update propagation theory
  - Interaction with scheduler
  - Merge tables, partition rollups, etc.
- Implemented correct algorithm
  - Has some scheduling restrictions
- Problem: scheduling restrictions cause update delays in RT tables.
  - Move to algorithm without scheduler restrictions.

Incremental Updates

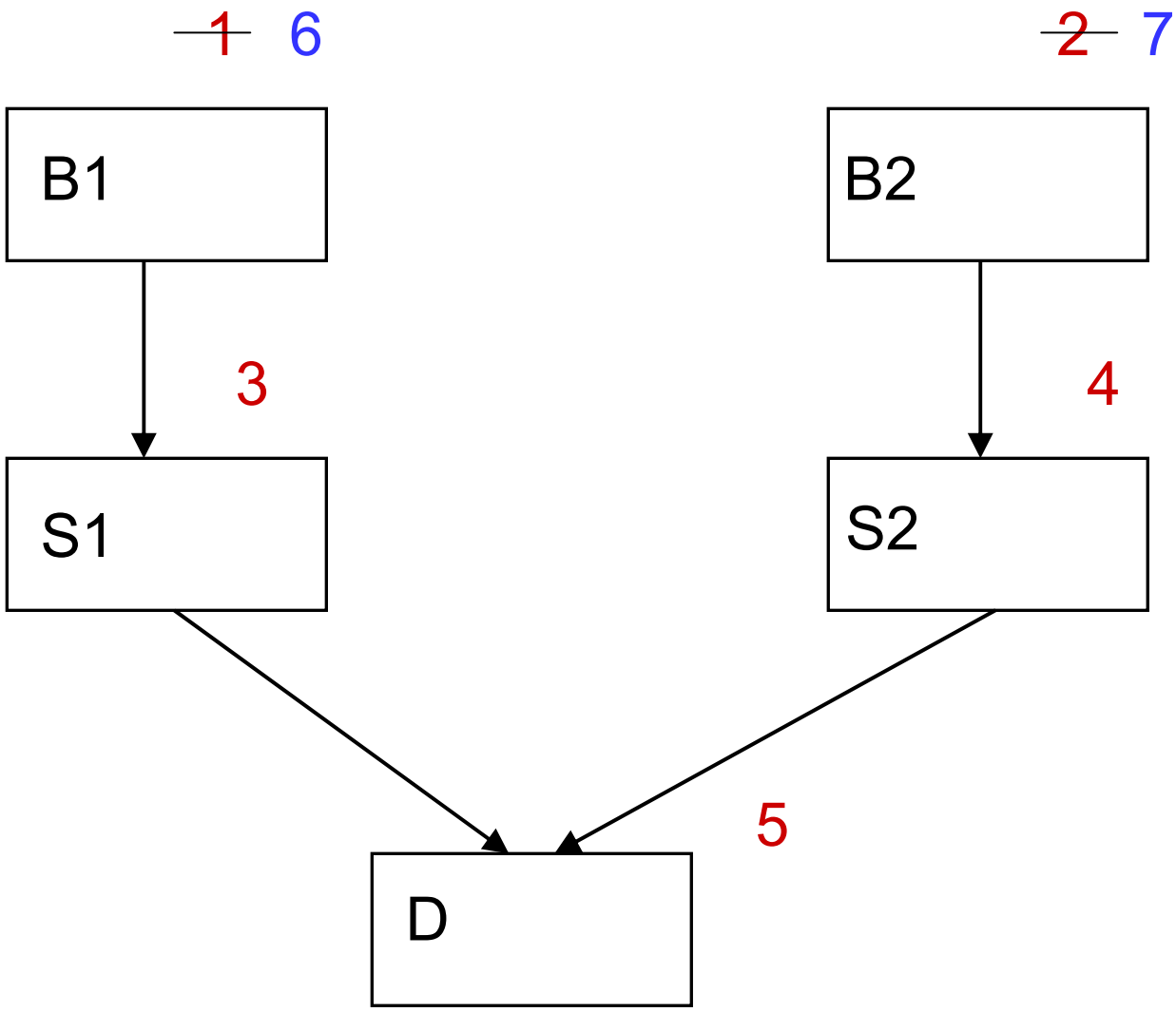


- Only propagate the increment.
- Update only those partitions whose sources have new data.
- How can we determine if a source partition has more recent data?

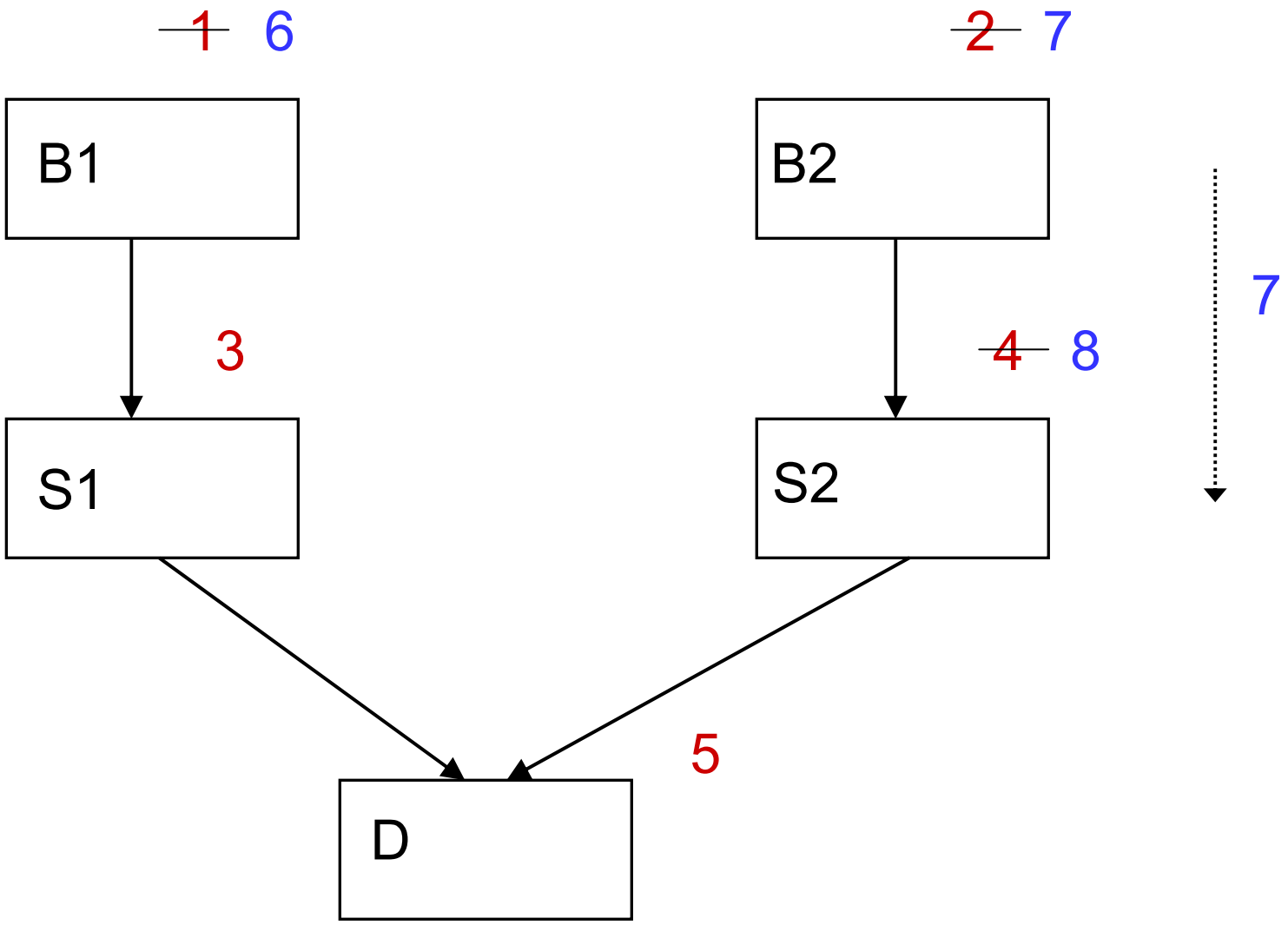
“make” doesn’t work



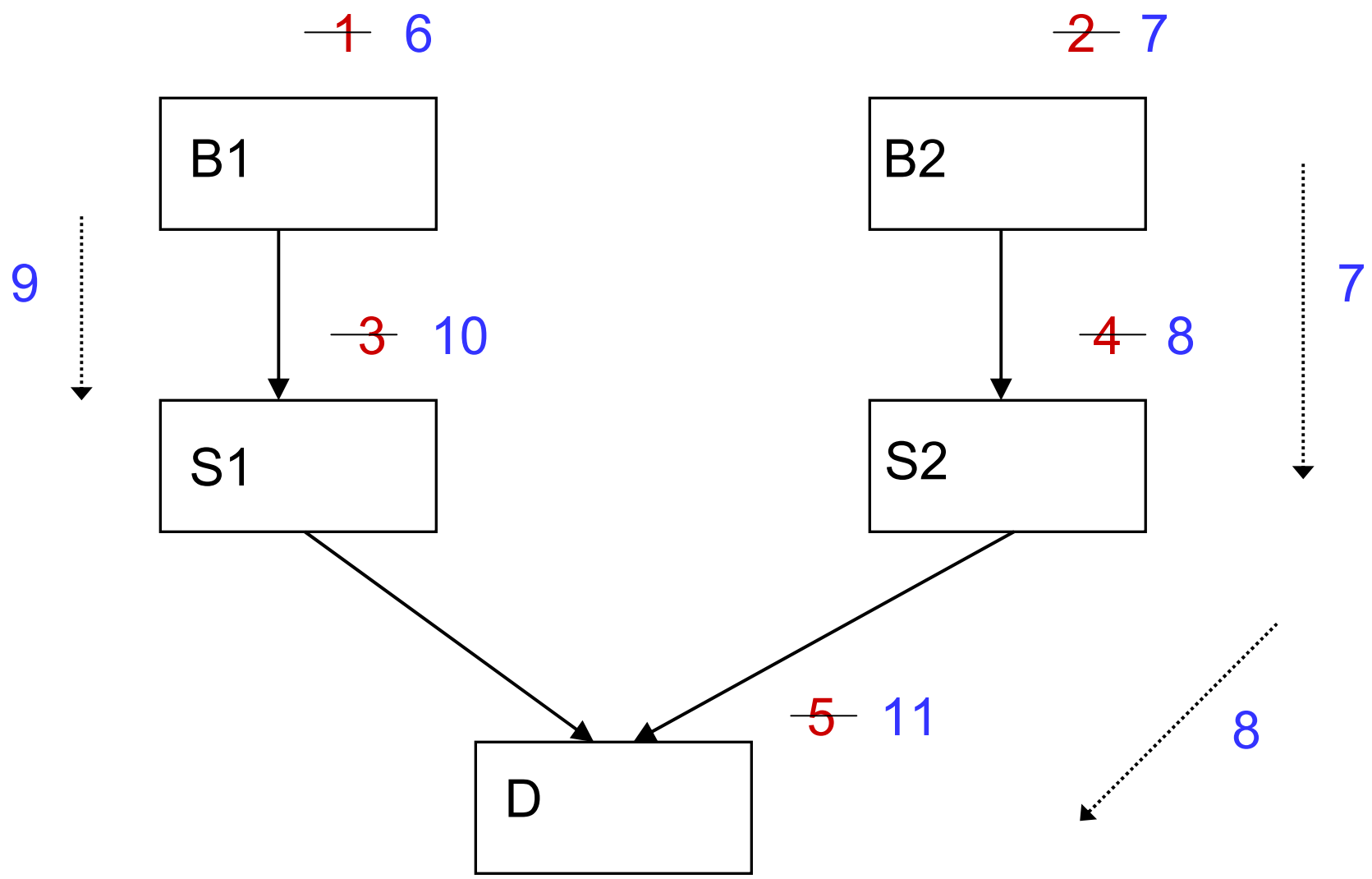
“make” doesn’t work



“make” doesn’t work

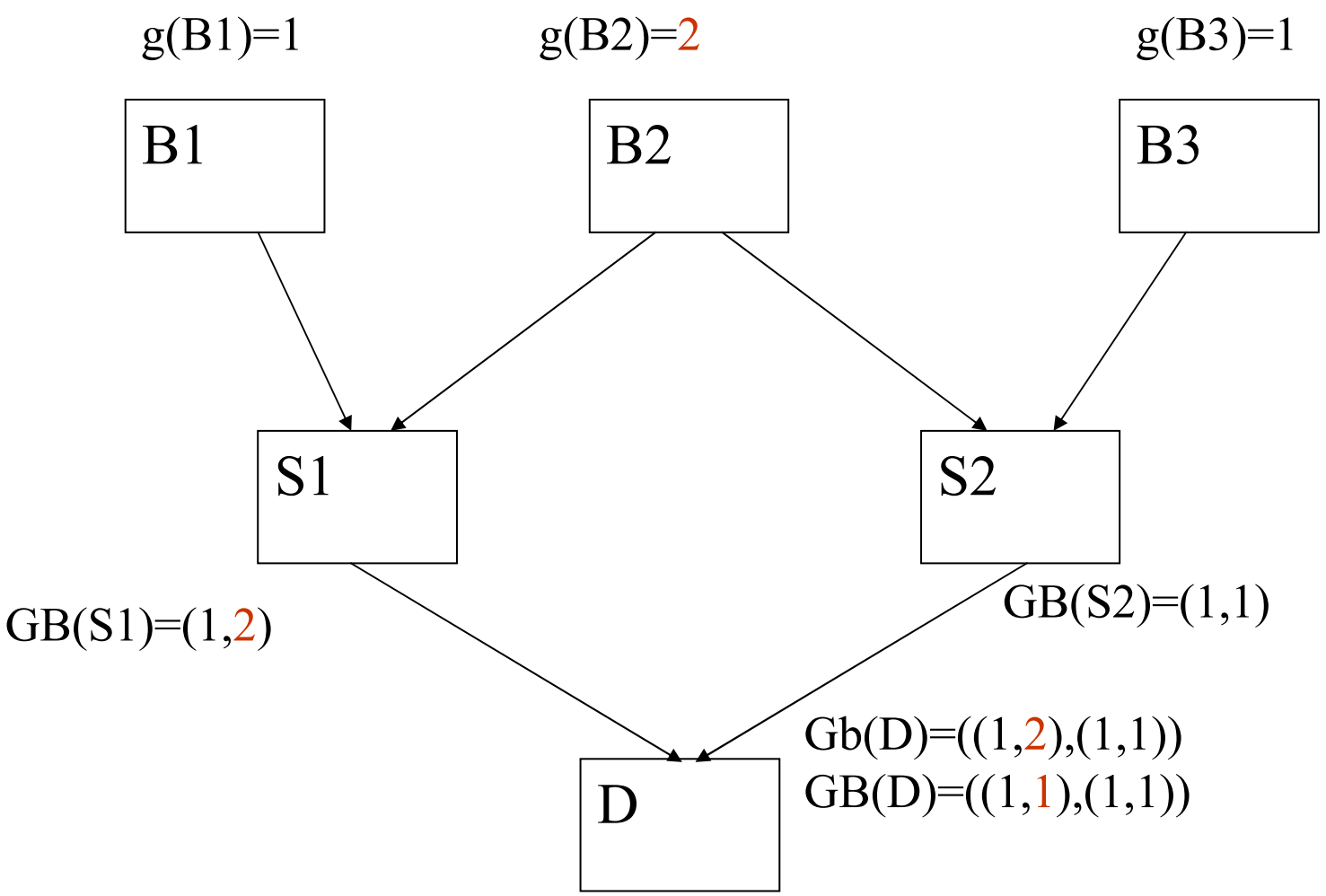


“make” doesn’t work



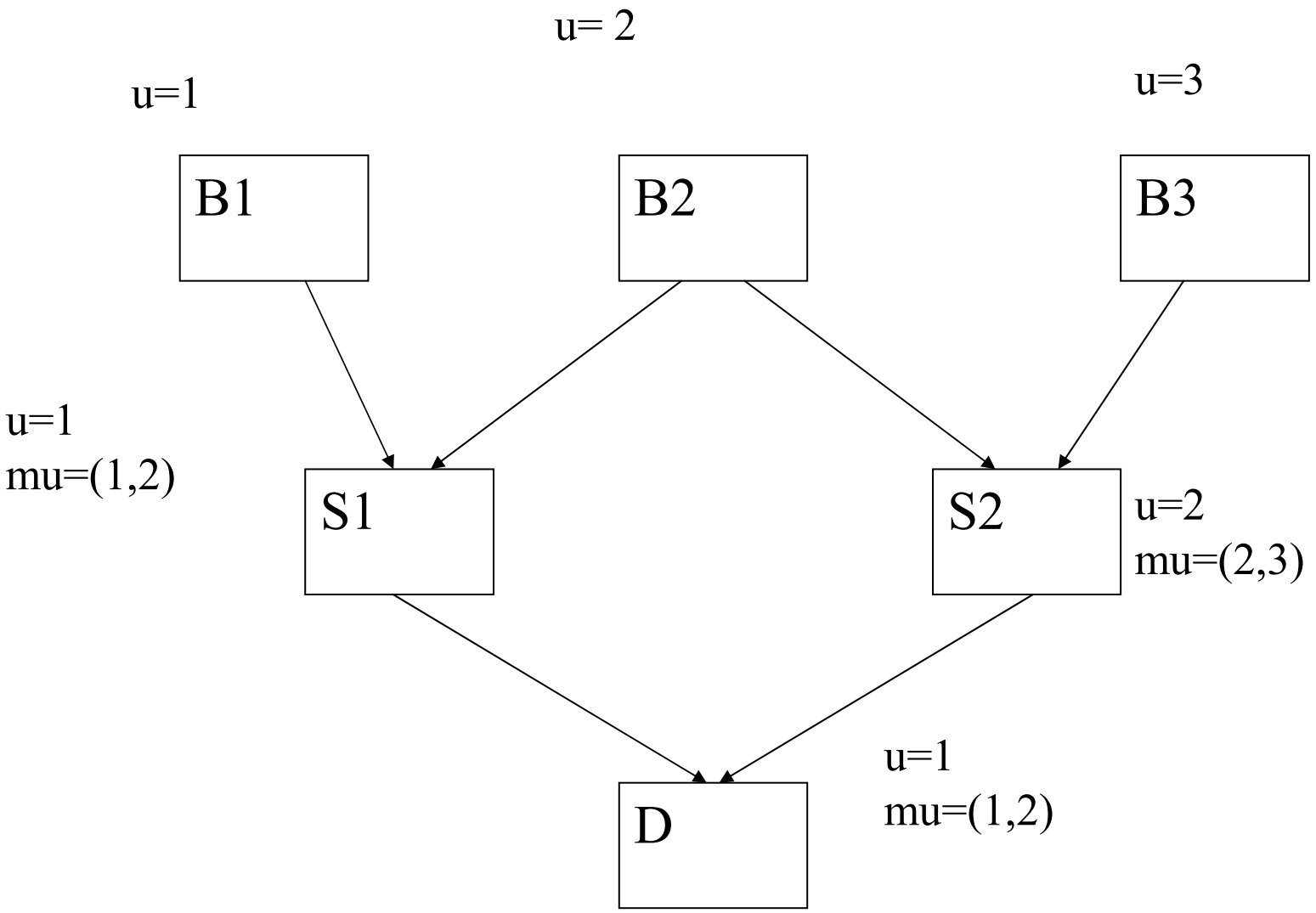
Correctness Theory

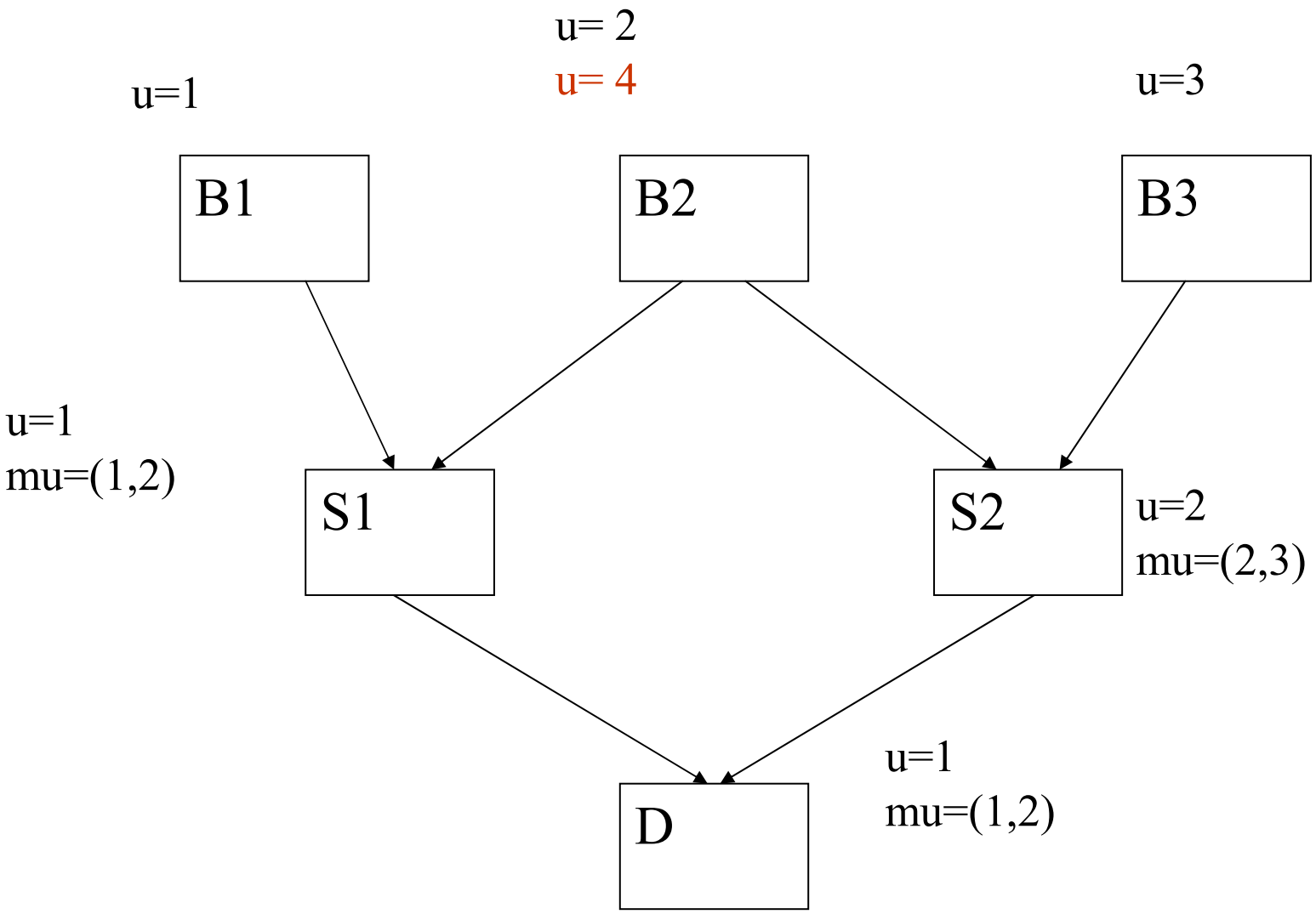
- The warehouse consists of *base* tables and *derived* tables.
- Each partition of a base table has a *generation number*: the number of times it has been updated.
- A derived table partition D depends on a collection B(D) of base table partitions
  - One entry for each distinct dependence path.
- GB(D) is the collection of generation numbers of the base table partitions in B(D) used to compute D.
- Gb(D) is the collection of base table partition numbers of D's source partitions
- Update D iff. Gb(D) > GB(D).

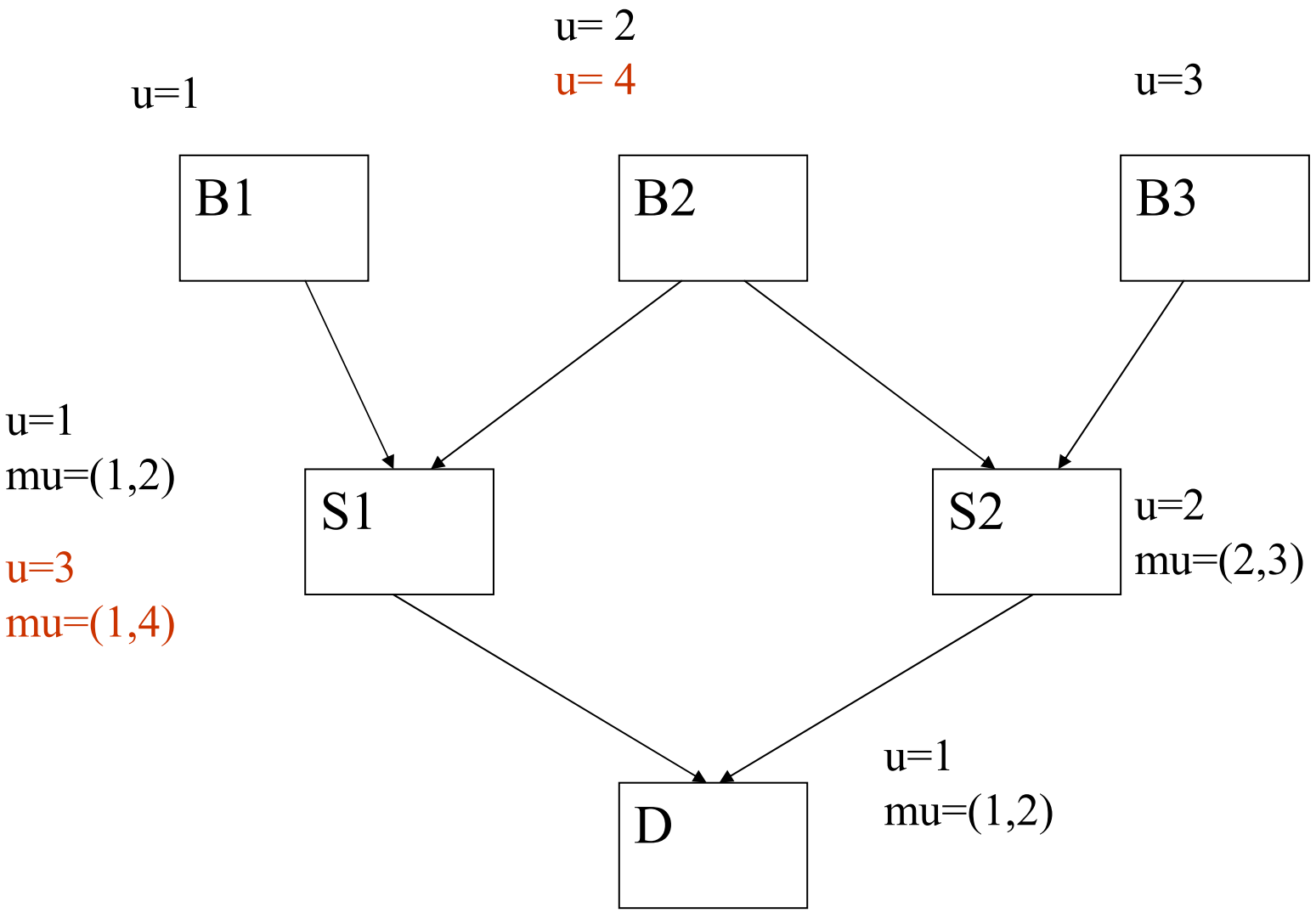

g(B1)=1
g(B2)=2
g(B3)=1
B1
B2
B3
S1
S2
GB(S1)=(1,2)
GB(S2)=(1,1)
Gb(D)=((1,2),(1,1))
GB(D)=((1,1),(1,1))
D

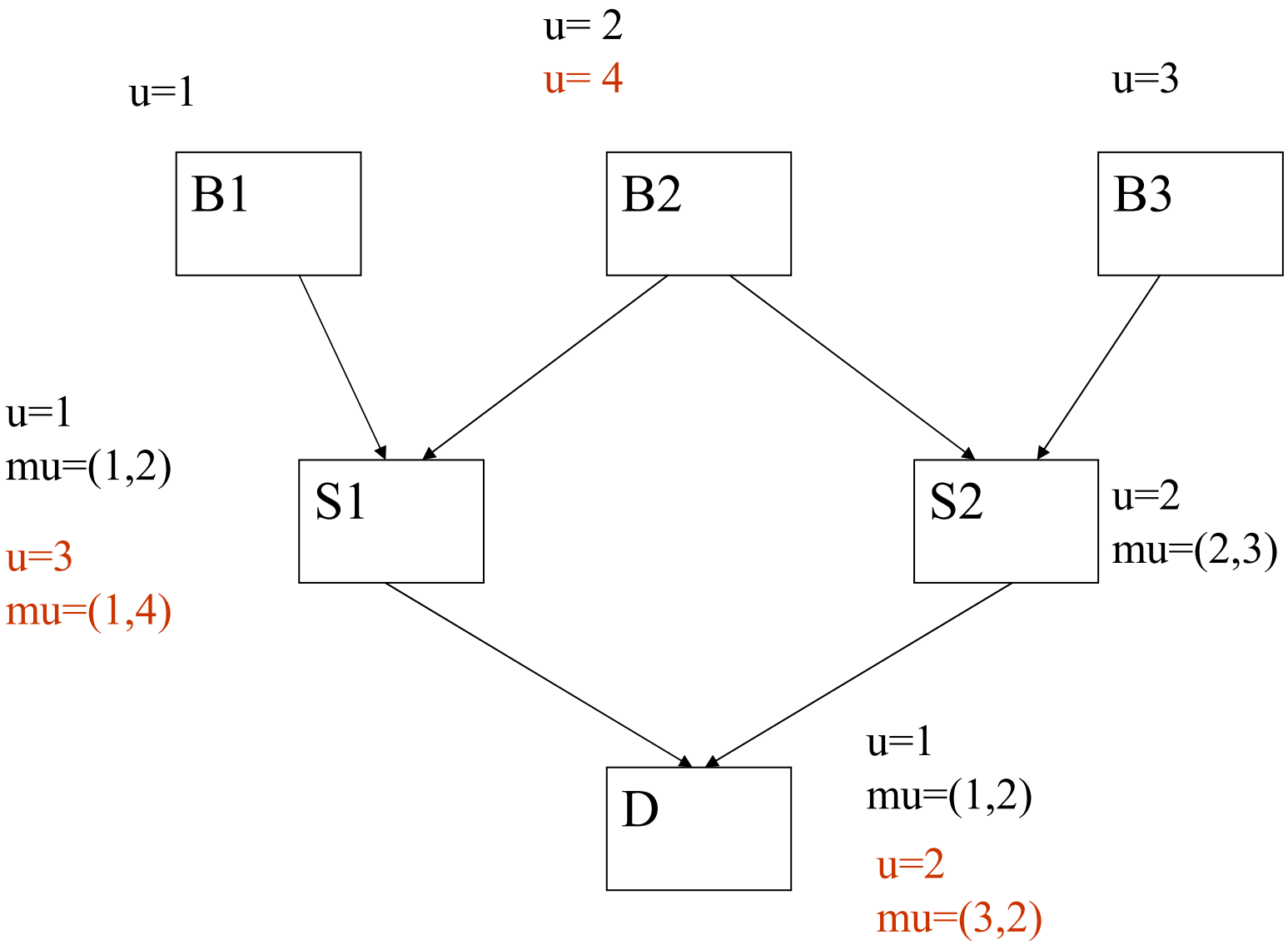
Source-vector Protocol

- Each table D maintains an update counter *ut*.
- Each partition d of D maintains an updatestamp *u* which is, assigned the value of *ut* when the partition is updated.
- Each partition also maintains a *source vector mu=(M1,..,Mn)*, which is the maximum updatestamp in the partitions from table Si that affect partition d.
- Update partition d iff. there is a source table Si with partitions (si1,..,sij) such that max(u(si1),…,u(sij)) > Mi.

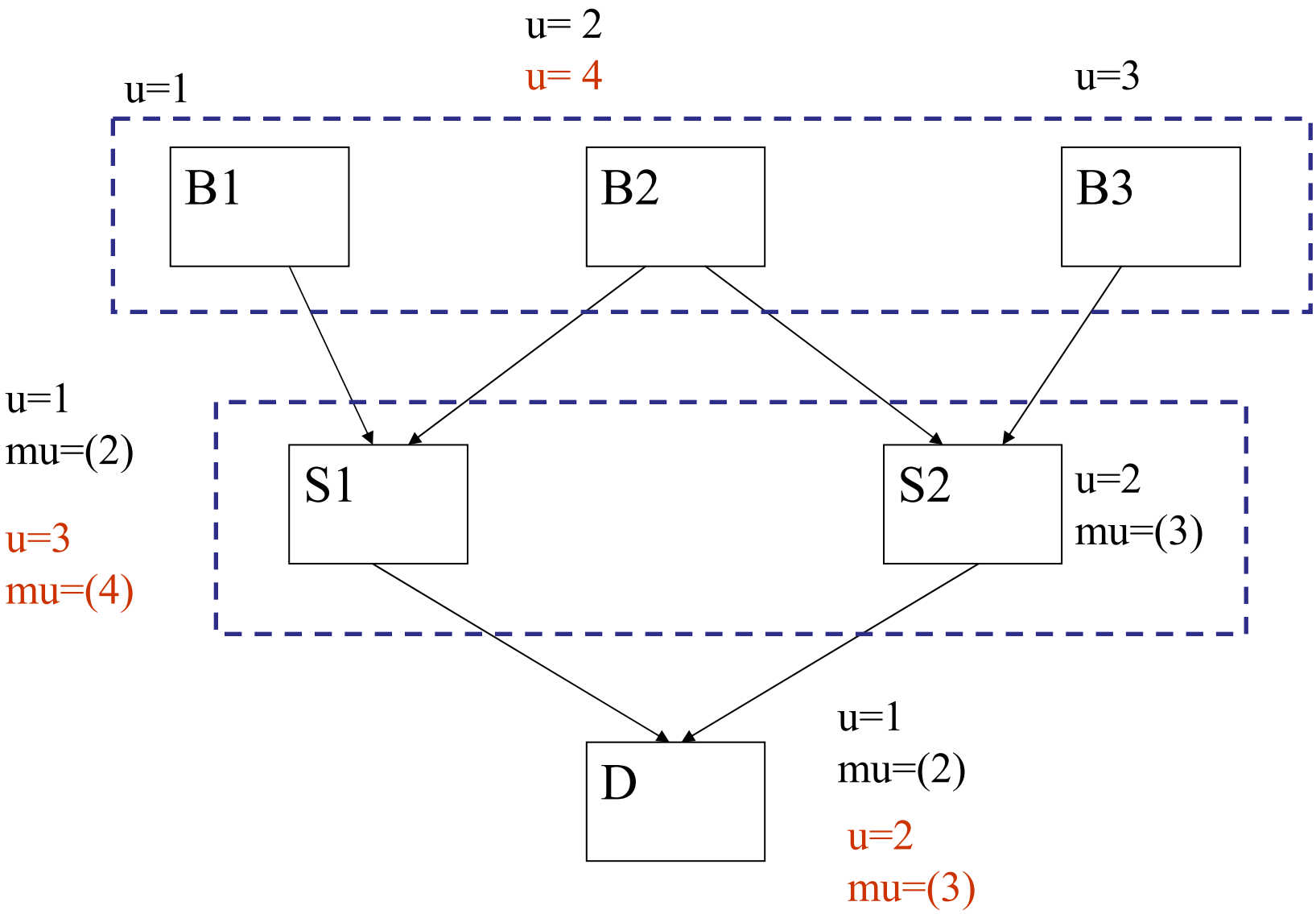

u= 2
u=1
u=3
B1
B2
B3
u=1
mu=(1,2)
S1
S2
u=2
mu=(2,3)
u=1
mu=(1,2)
D


u= 2
u= 4
u=1
u=3
B1
B2
B3
u=1
mu=(1,2)
S1
S2
u=2
mu=(2,3)
u=1
mu=(1,2)
D


u= 2
u= 4
u=1
u=3
B1
B2
B3
u=1
mu=(1,2)
u=3
mu=(1,4)
S1
S2
u=2
mu=(2,3)
u=1
mu=(1,2)
D


u= 2
u= 4
u=1
u=3
B1
B2
B3
u=1
mu=(1,2)
u=3
mu=(1,4)
S1
S2
u=2
mu=(2,3)
u=1
mu=(1,2)
u=2
mu=(3,2)
D

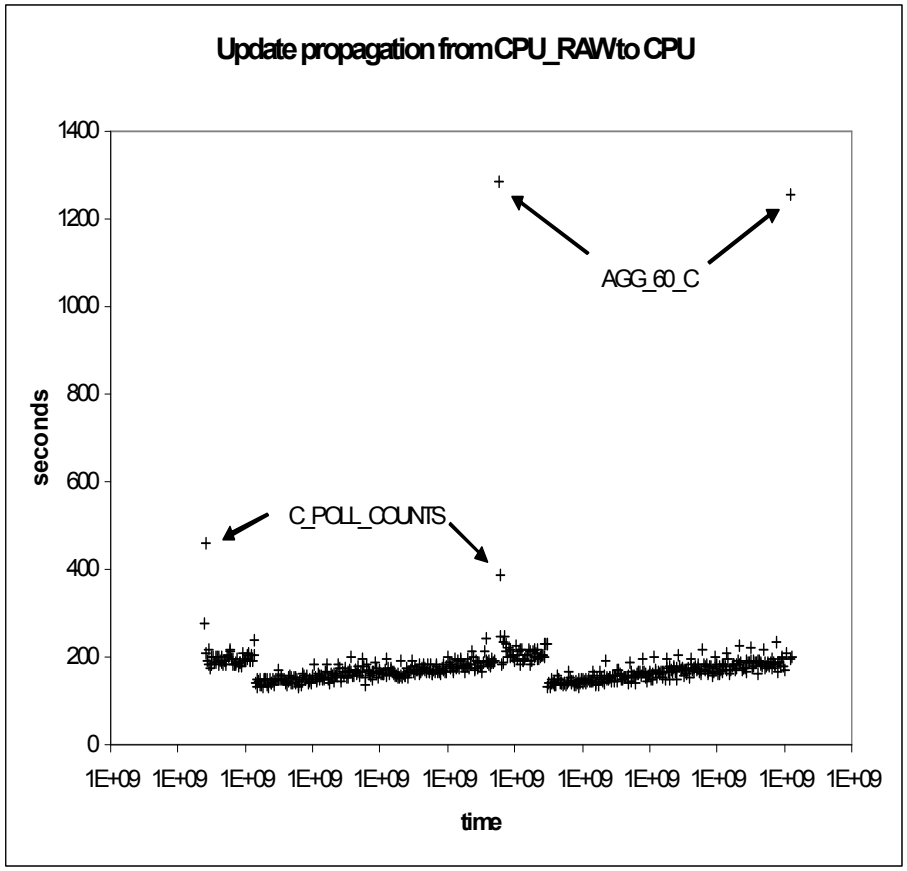
Suppose B1, B2, B3 are in table B, S1, S2 in table S



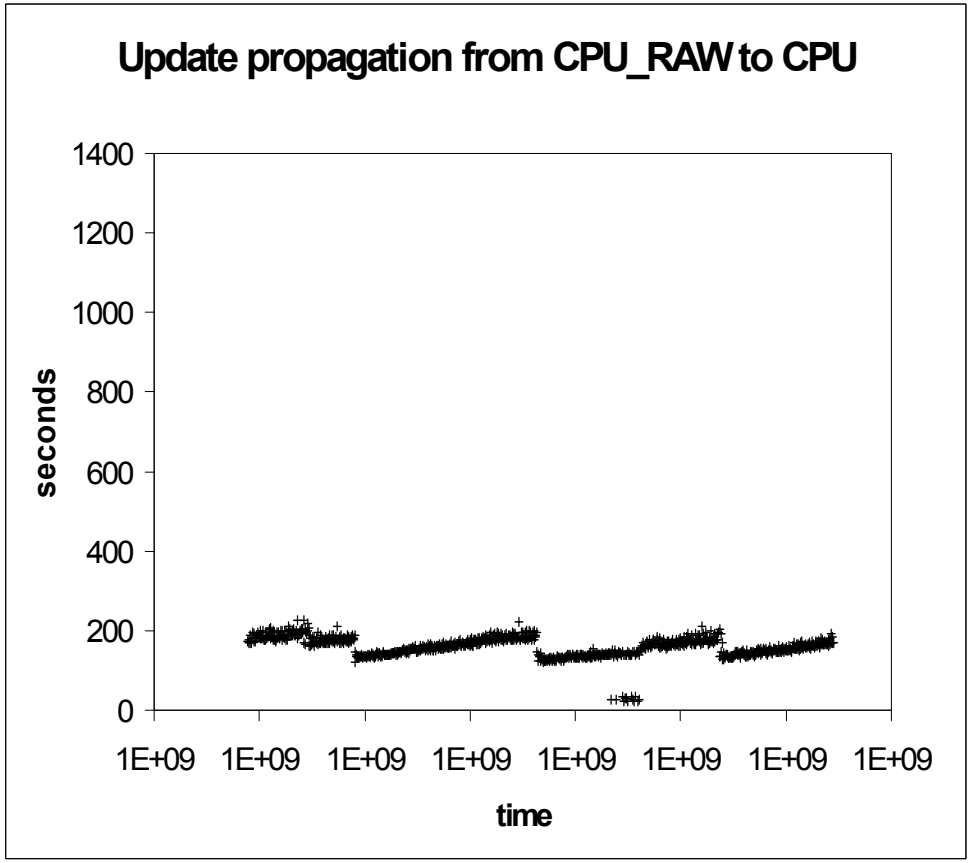
Extensions

- Interval-timestamp protocol
  - Store start and end timestamp of the update.
  - Fixed-size metadata, but it requires synchronized timestamps in a cluster
- Trailing-edge consistency
  - Propagate “no more updates” punctuation
- MERGE tables (Union)
  - Treat new partitions as newly loaded.
- Partition Rollups
  - Problem: load data once per minute, store for 2 years
  - 1-minute partitions => 1,051,200 partitions to cover a year
  - Rollup older partitions
    - First 2 days : 1 minute partitions
    - Next 728 days : 1 day partitions
    - 3608 partitions to cover 2 years, but with efficient real-time updates.
- Any single-timestamp protocol requires scheduling restrictions
  - Scheduling restrictions are bad for a real-time warehouse.

Effects of Scheduling Restrictions



Starting-timestamp update protocol



Interval-timestamp update protocol

Conclusions

- Update propagation is a critical component of an on-line stream warehouse.
  - Continual loading
  - Thousands of tables.
- `Make` algorithm doesn't work.
- Best algorithm uses a modified vector timestamp.
  - Aggressive timestamp compression.
- Critical component of many stream warehouses in use in AT&T.