Hash History:
A Method for Reconciling Mutual Inconsistency in Optimistic Replication

Brent ByungHoon Kang,
Robert Wilensky and John Kubiatowicz

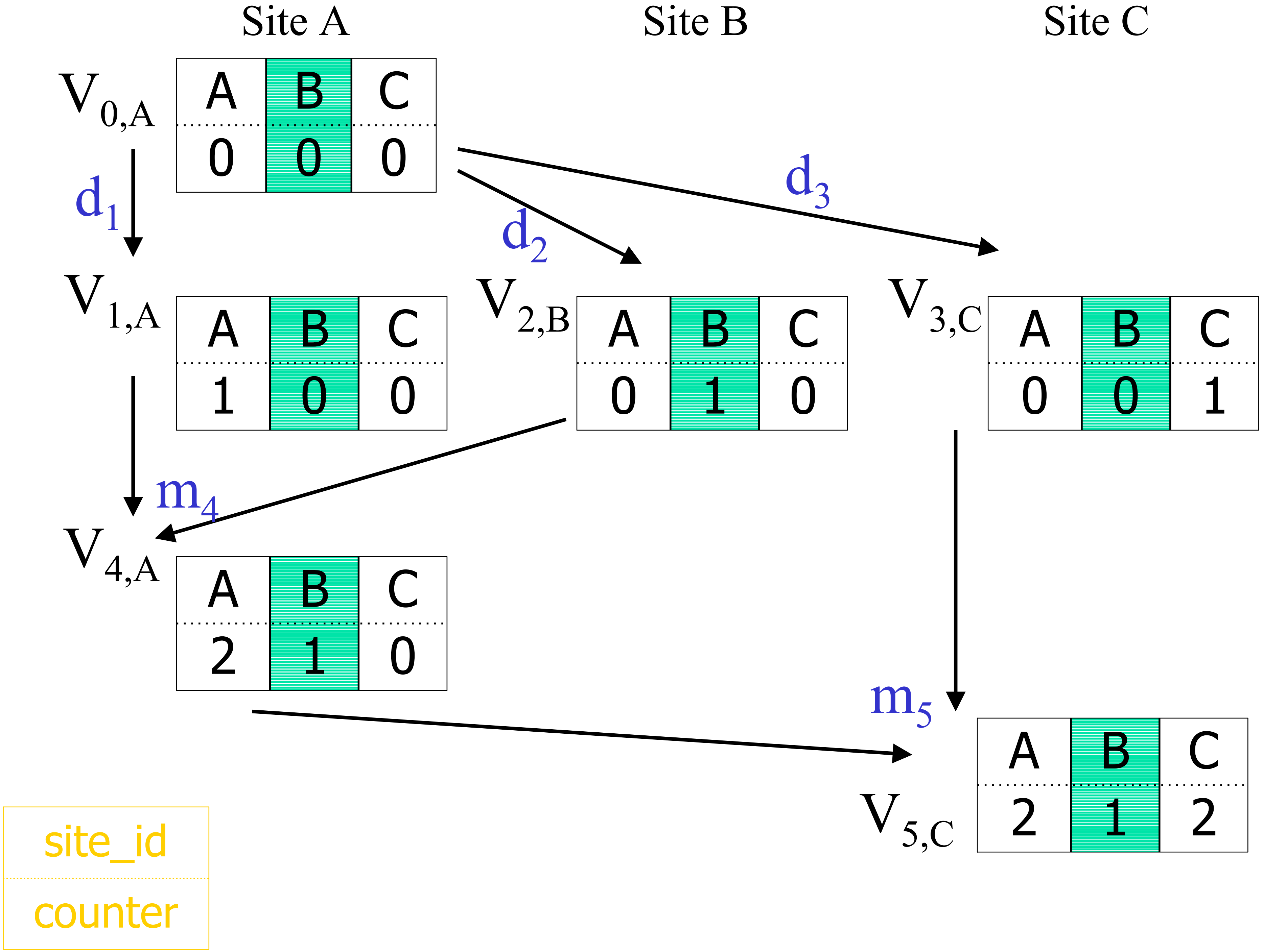
CS Division, UC Berkeley

Background

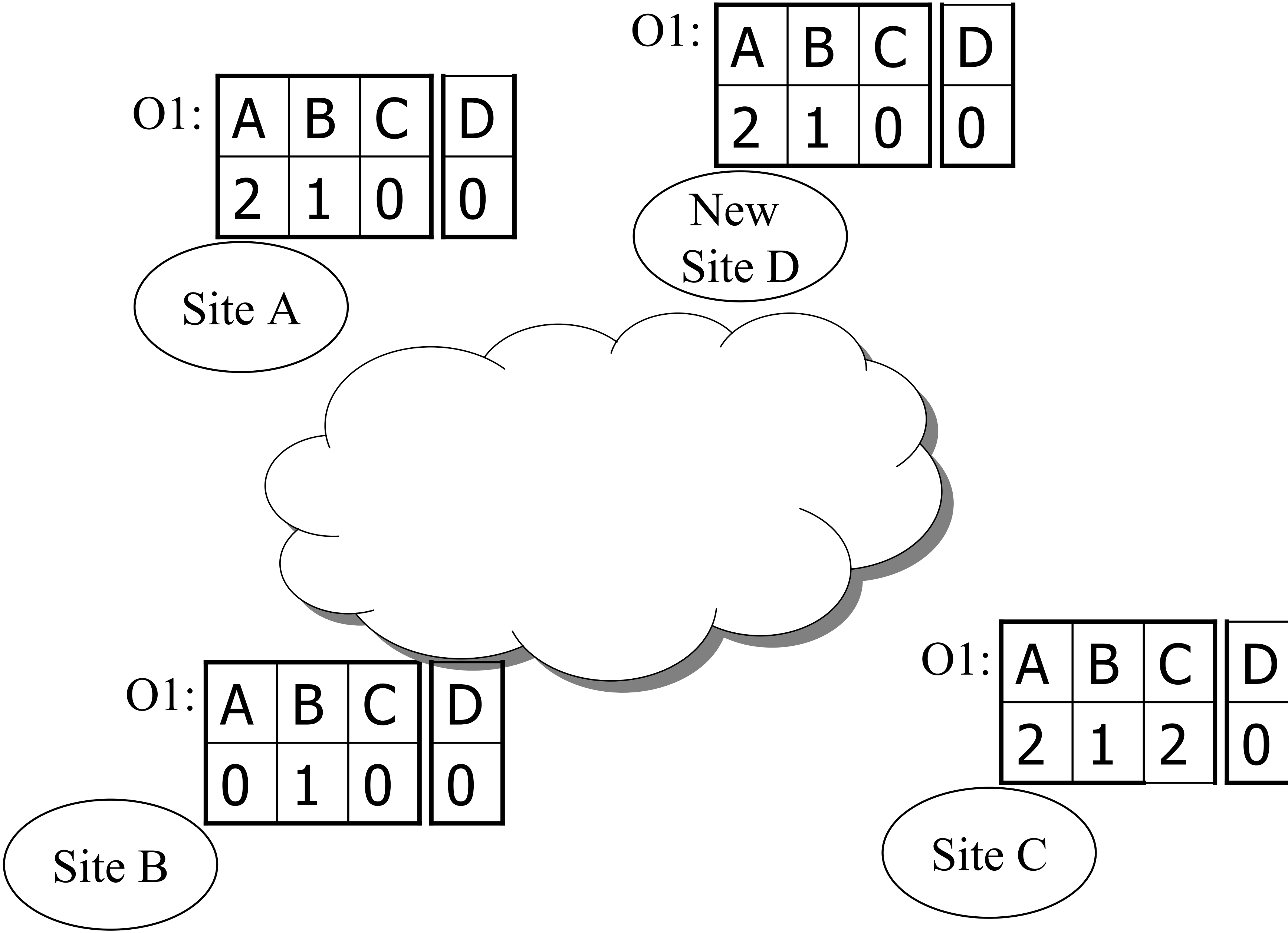
- Optimistic Replication
  - Allow mutable replica to be inconsistent temporarily
    - in a controlled way
    - for high availability and performance
  - Tentative update support in OceanStore
  - Bayou, USENET, and Peer-to-Peer File System (e.g., Ivy, Pangaea, etc.)
- Need Mechanism for
  - Figuring out the ordering among updates
  - Extracting deltas to be exchanged during reconciliation

Previous Approaches: Version Vectors

- Widely used in reconciling replicas
  - In most weakly consistent replication systems
  - Bayou, Ficus, Coda, Ivy, Pangaea … etc.
- Complexity of management grows
  - As new replica site added or deleted
  - Need to assign unique id dynamically for newly added replica sites
- Doesn’t scale as number of replica site increases
  - Version vector needs one entry for each replica site
  - Size of vector grows in proportion to number of replica sites

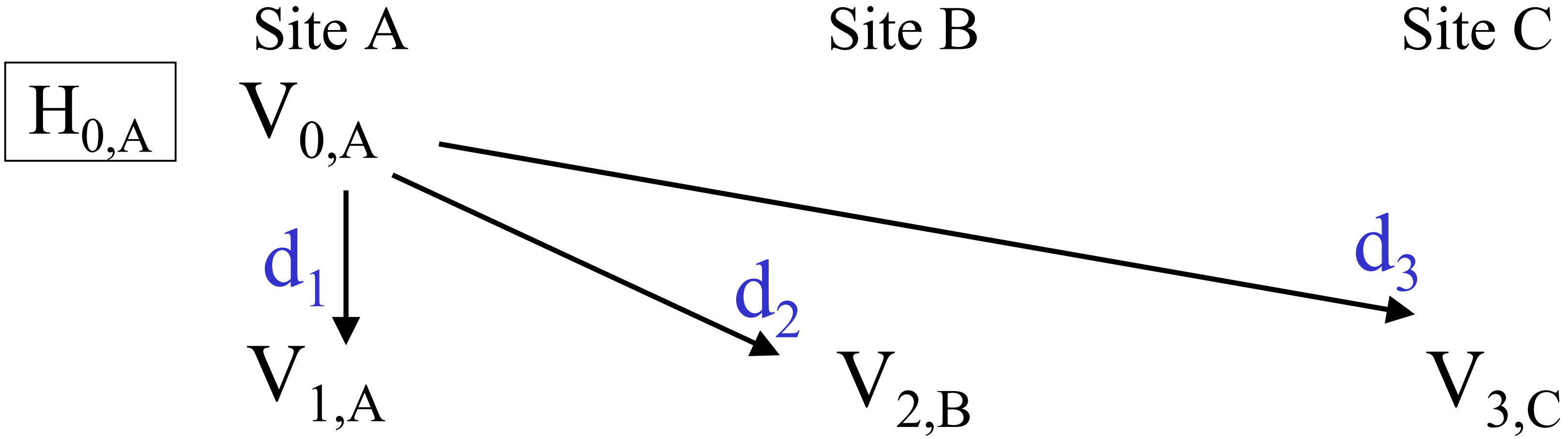


Site A
Site B
Site C
V0,A
A B C
0 0 0
d1
d2
d3
V1,A
A B C
1 0 0
V2,B
A B C
0 1 0
V3,C
A B C
0 0 1
m4
V4,A
A B C
2 1 0
m5
V5,C
A B C
2 1 2
site_id
counter

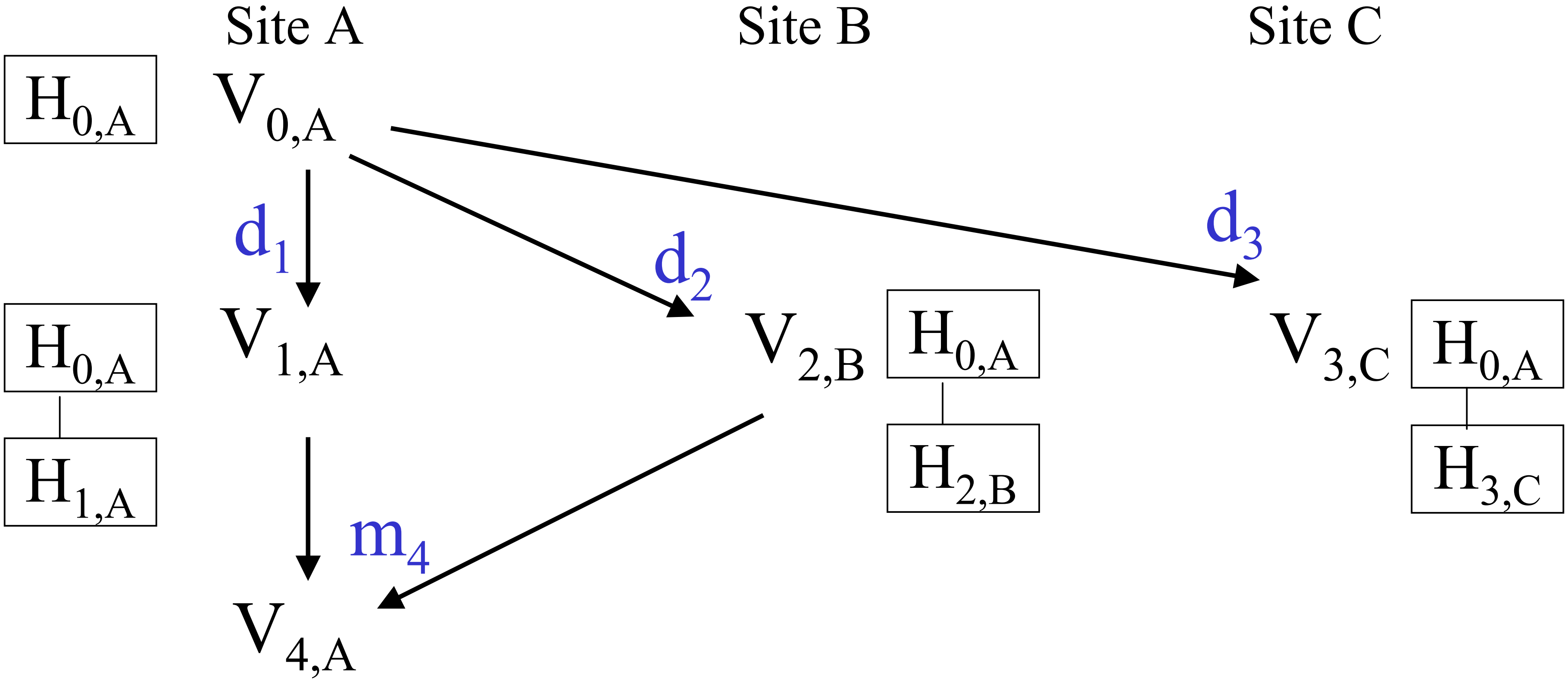

O1:
A B C D
2 1 0 0
Site A
O1:
A B C D
2 1 0 0
New
Site D
O1:
A B C D
0 1 0 0
Site B
O1:
A B C D
2 1 2 0
Site C

Our Proposal: Hash History

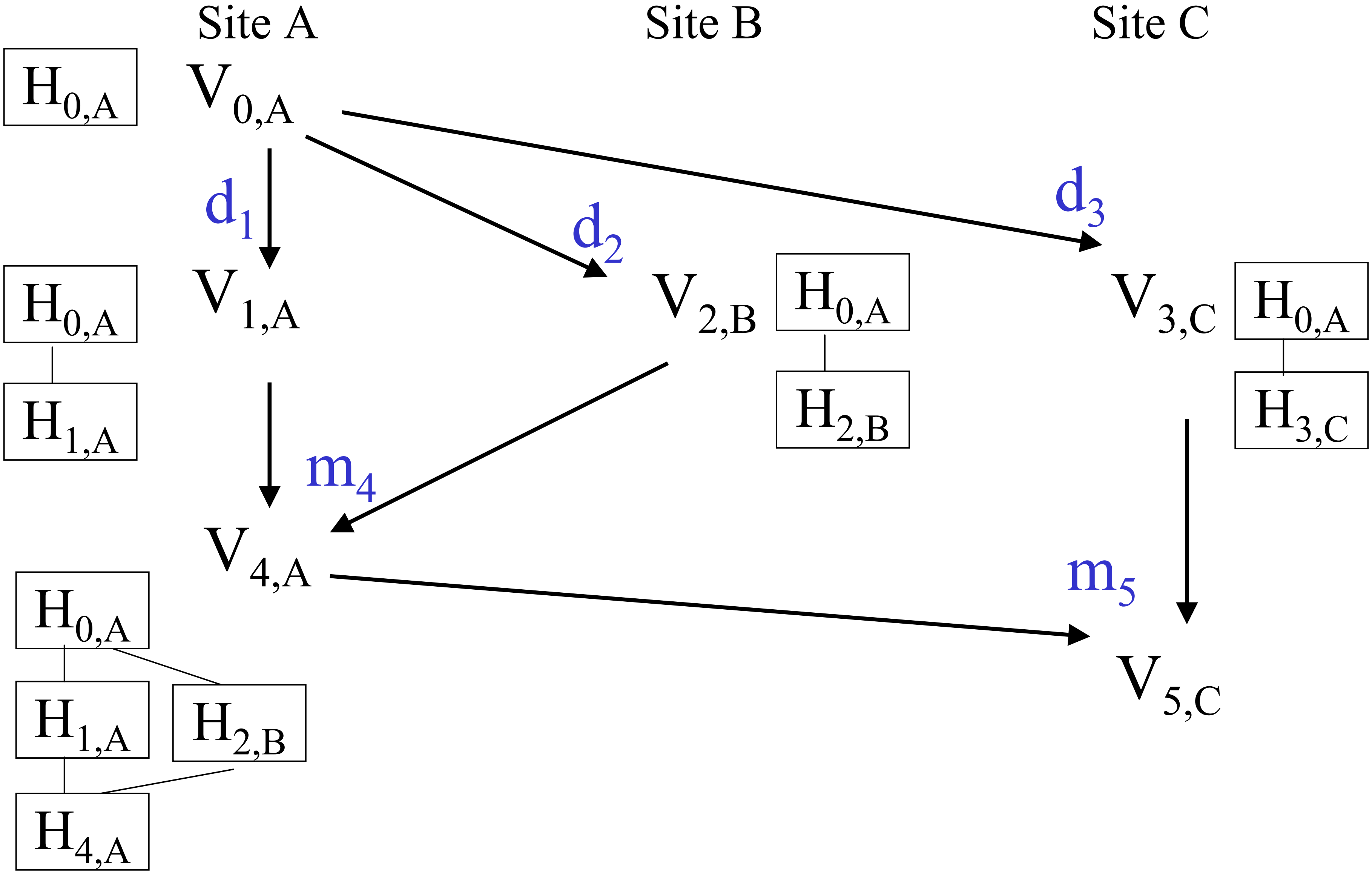
- Each site keeps a record of the hash of each version
  - Capture dependency among versions as a directed graph of version hashes (i.e., hash history)
- The sites exchange the hash history in reconciling replicas
- The most recent common ancestral version can be found, if no version dominates
  - Useful hints in a subsequent diffing/merging



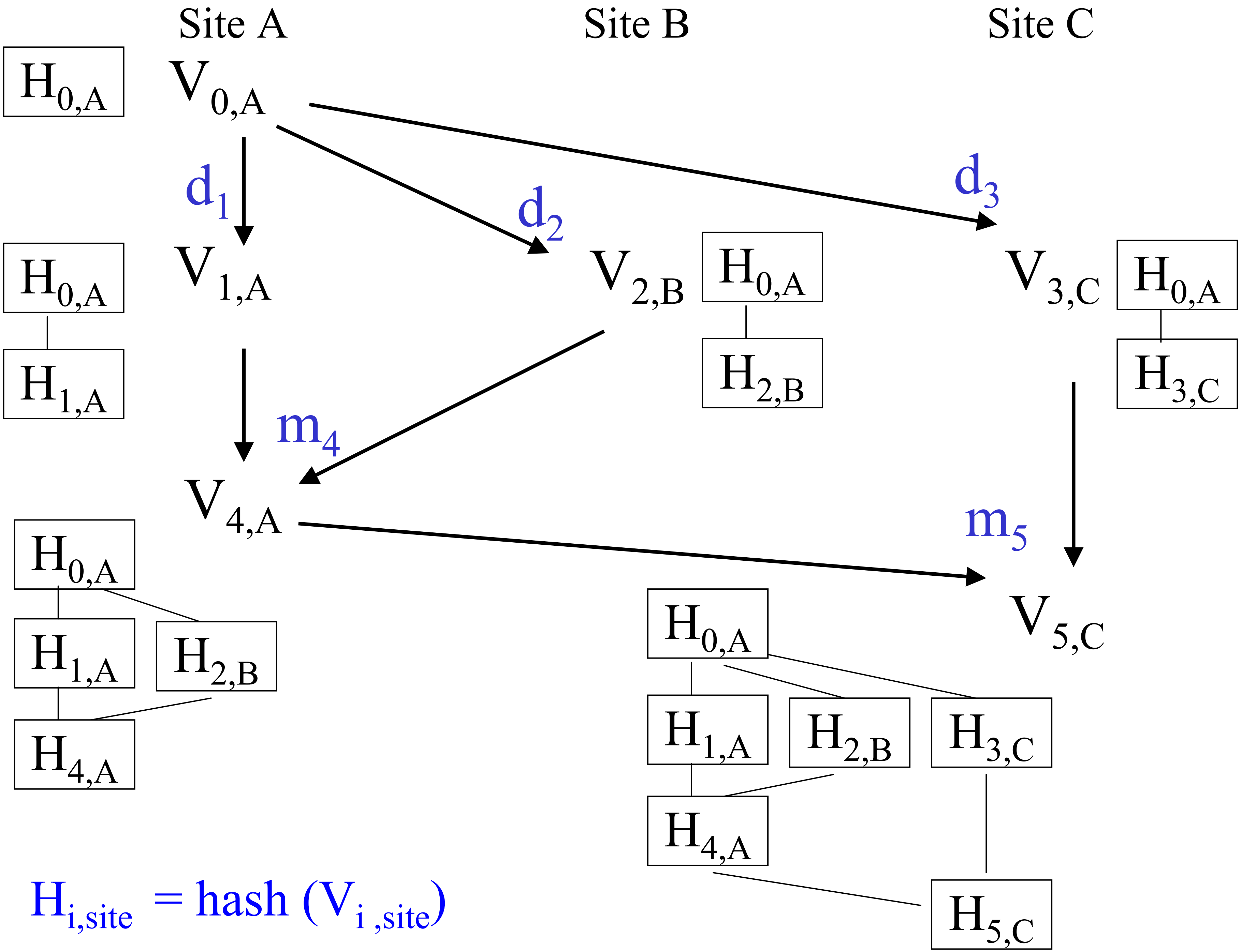
$H_{i,site} = hash\ (V_{i\ ,site})$



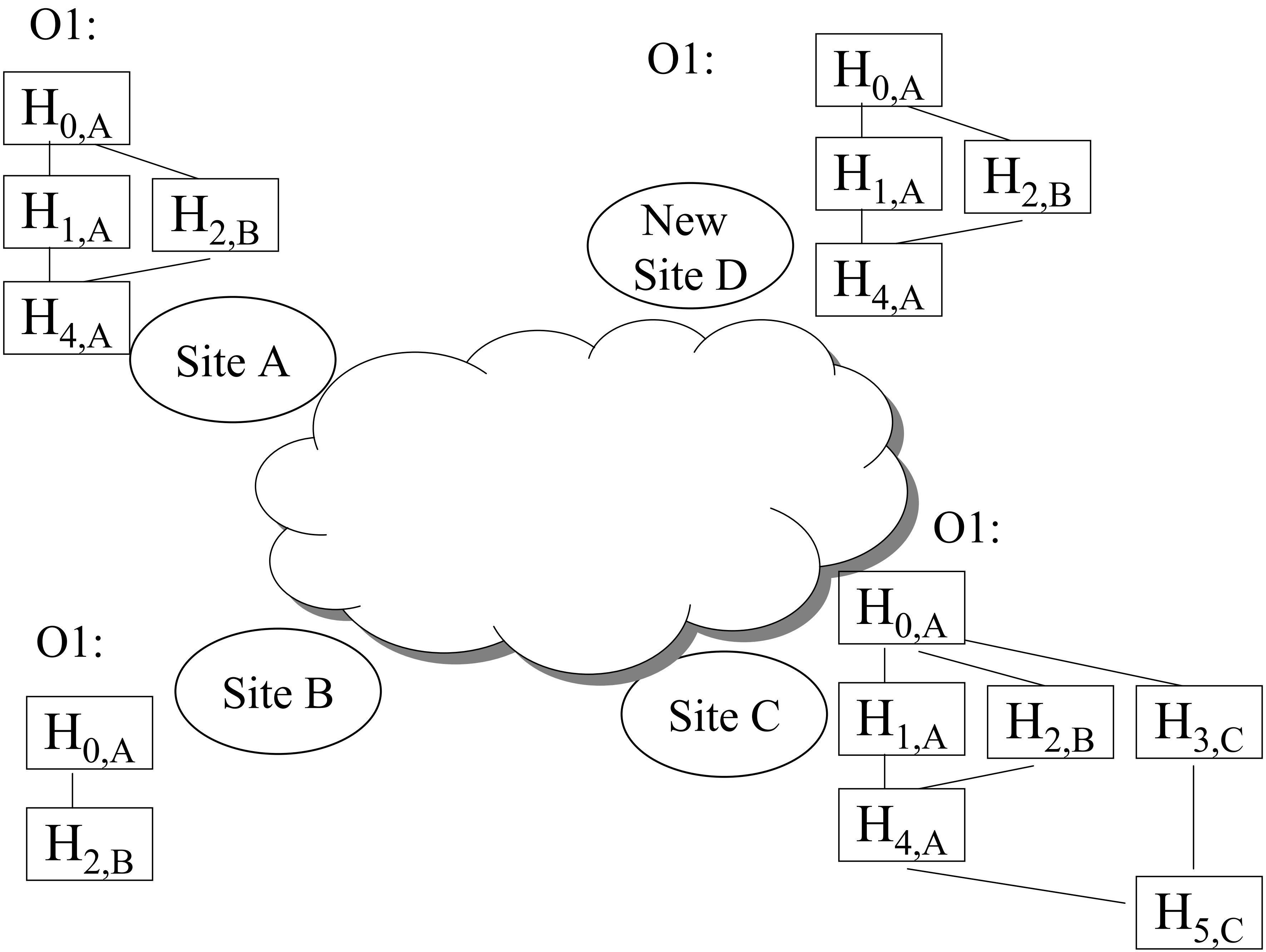
$H_{i,site} = hash\ (V_{i\ ,site})$



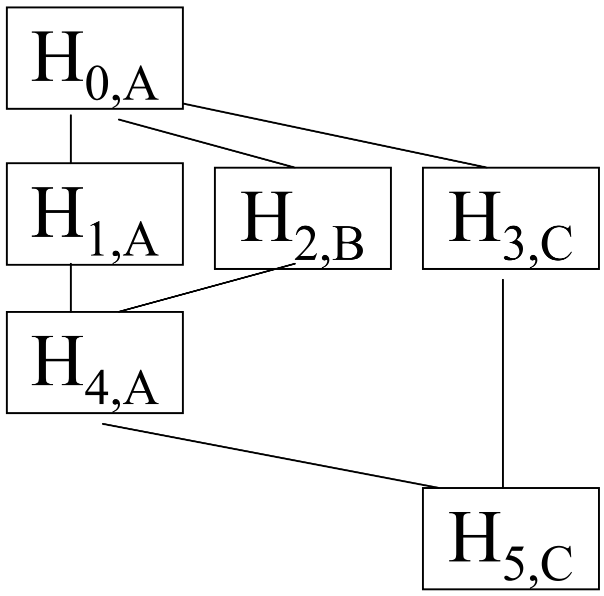
$$H_{i,site} = \text{hash}\,(V_{i\,,site})$$



$H_{i,site} = hash(V_{i,site})$


O1:
$H_{0,A}$
$H_{1,A}$
$H_{2,B}$
$H_{4,A}$
Site A
O1:
$H_{0,A}$
$H_{1,A}$
$H_{2,B}$
$H_{4,A}$
New Site D
O1:
$H_{0,A}$
$H_{2,B}$
Site B
Site C
O1:
$H_{0,A}$
$H_{1,A}$
$H_{2,B}$
$H_{3,C}$
$H_{4,A}$
$H_{5,C}$

Hash History Graph



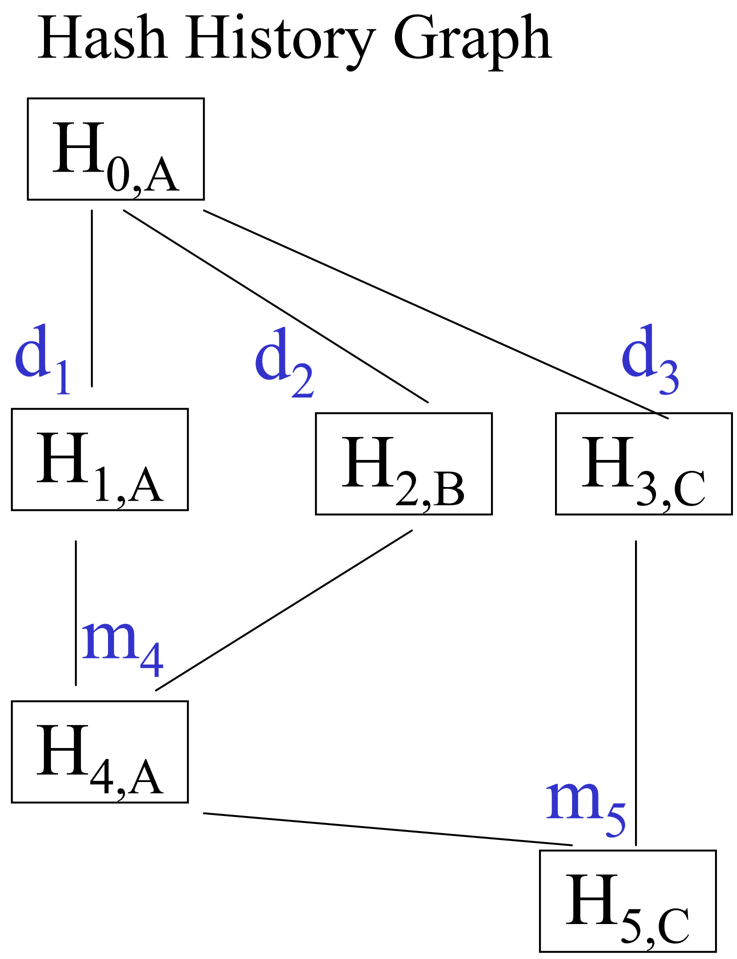
(a)

Hash History with Hashtable

Child	Parents
$H_{0,A}$	null
$H_{1,A}$	$H_{0,A}$
$H_{2,B}$	$H_{0,A}$
$H_{3,C}$	$H_{0,A}$
$H_{4,A}$	$H_{1,A}$: $H_{2,B}$
$H_{5,C}$	$H_{4,A}$: $H_{3,C}$

Latest : $H_{5,C}$

(b)



(a)

Hash History with Hashtable

Child	Parents	delta
$H_{0,A}$	null	null
$H_{1,A}$	$H_{0,A}$	d_1
$H_{2,B}$	$H_{0,A}$	d_2
$H_{3,C}$	$H_{0,A}$	d_3
$H_{4,A}$	$H_{1,A}$: $H_{2,B}$	m_4
$H_{5,C}$	$H_{4,A}$: $H_{3,C}$	m_5

Latest : $H_{5,C}$

(b)

HH Properties

- Size of hash history is unbounded
  - Simple Aging
  - Sharable Archived Hash Histories
- Can capture equality case
  - When two different schedule of deltas produce the same output
  - Helps faster convergence

Why Less Conflict in HH than VV

- HH can covey equality information to the descendents while VV cannot
  - E.g., v1 = <A:4,B:5,C:0,D:0,E:0,F:0>
  - v2 = <A:5,B:4,C:0,D:0,E:0,F:0>
  - C merges then v3 = <A:5,B:5,C:1,D:0,E:0,F:0>
  - E merges then v4 = <A:5,B:5,C:0,D:0,E:1,F:0>
  - v3 and v4 could be the same but VV shows conflict !
- If v3 and v4 are considered equal, then
  - all descendents of v4 will dominate v3.
- If v3 and v4 are considered as in conflict,
  - all descendents of v4, will be in conflict with v3

Experiment Goal

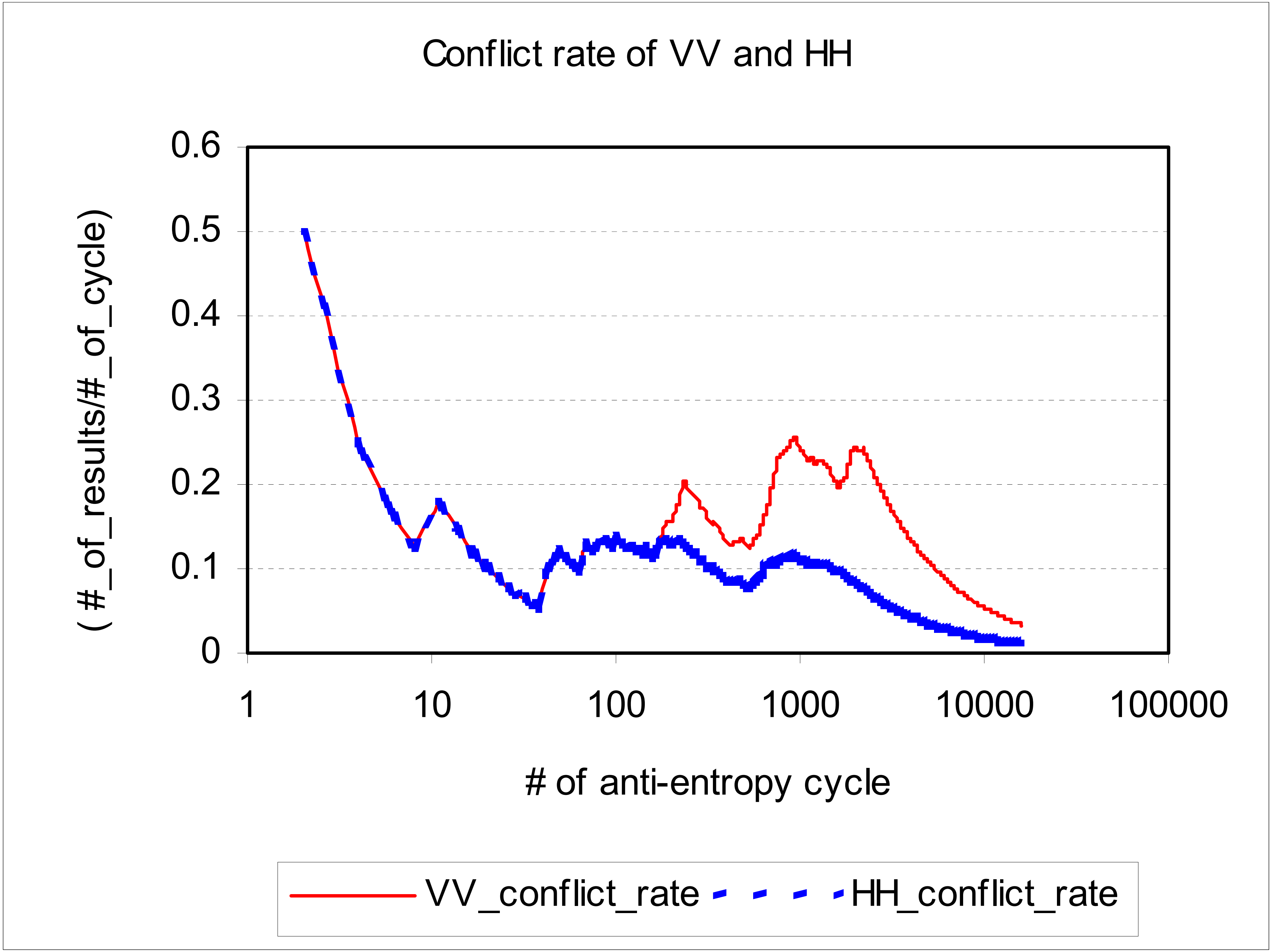
- Comparison with version vector result:
  - HH converges faster with a lower conflict rate than a version vector scheme
  - To what extent is this true in practice?
- Aging Policy:
  - the aging period for pruning hash history
  - vs. HH size
  - vs. the false conflict rate due to aging
    - when the pruned part of the hash history is required for determining the version dominance

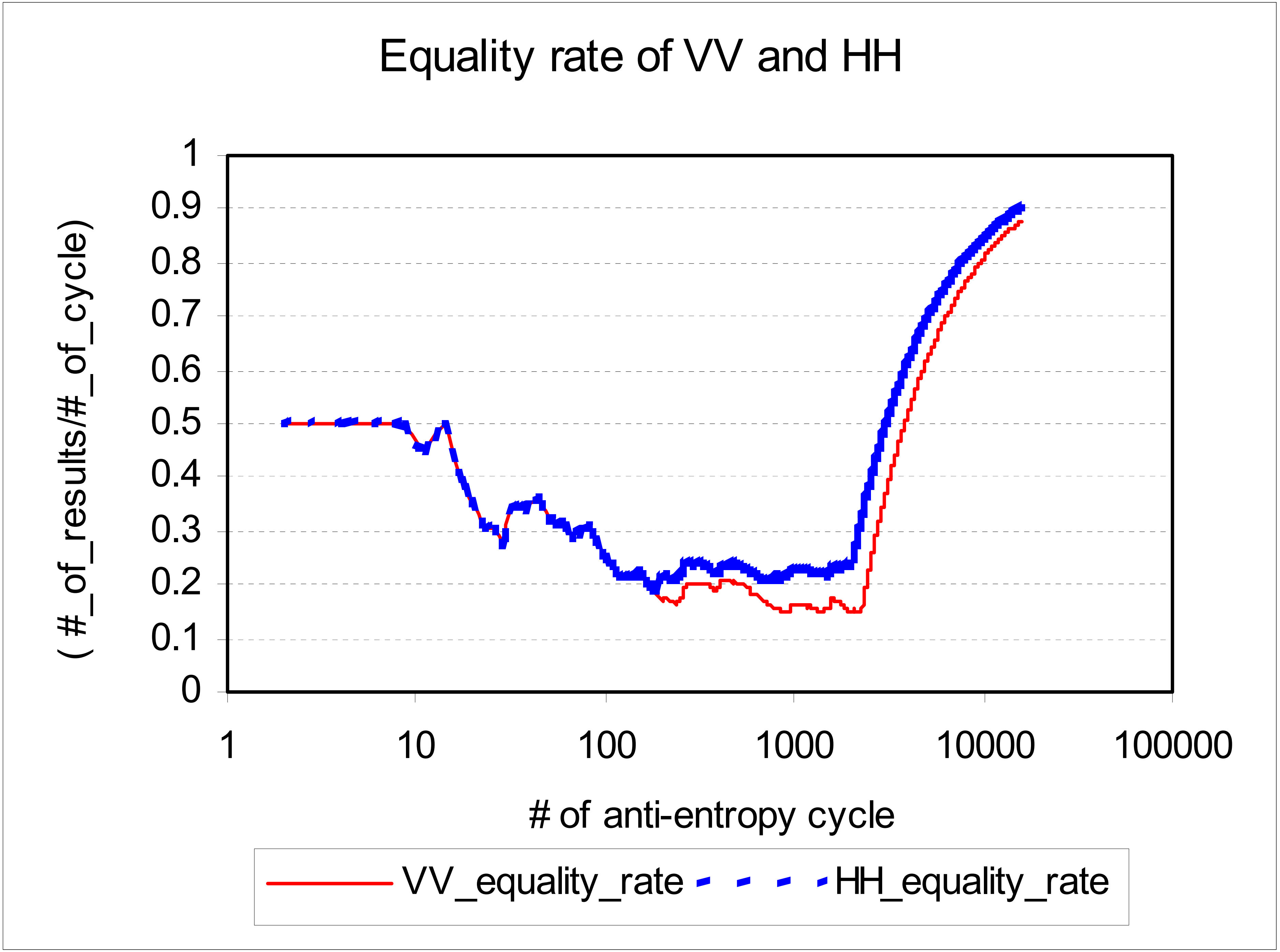
Simulation Setup

- Event-driven simulator
  - Events are collected from CVS logs
  - Each user represent a replica site
  - Reads the event <time, user, filename>
  - After each event, the simulator
    - repeats the anti-entropy for 50% (or 25%) of the total number of sites.
    - E.g., if there are 20 sites so far, the anti-entropy is repeated for 10 times with 50% parameter after each event.

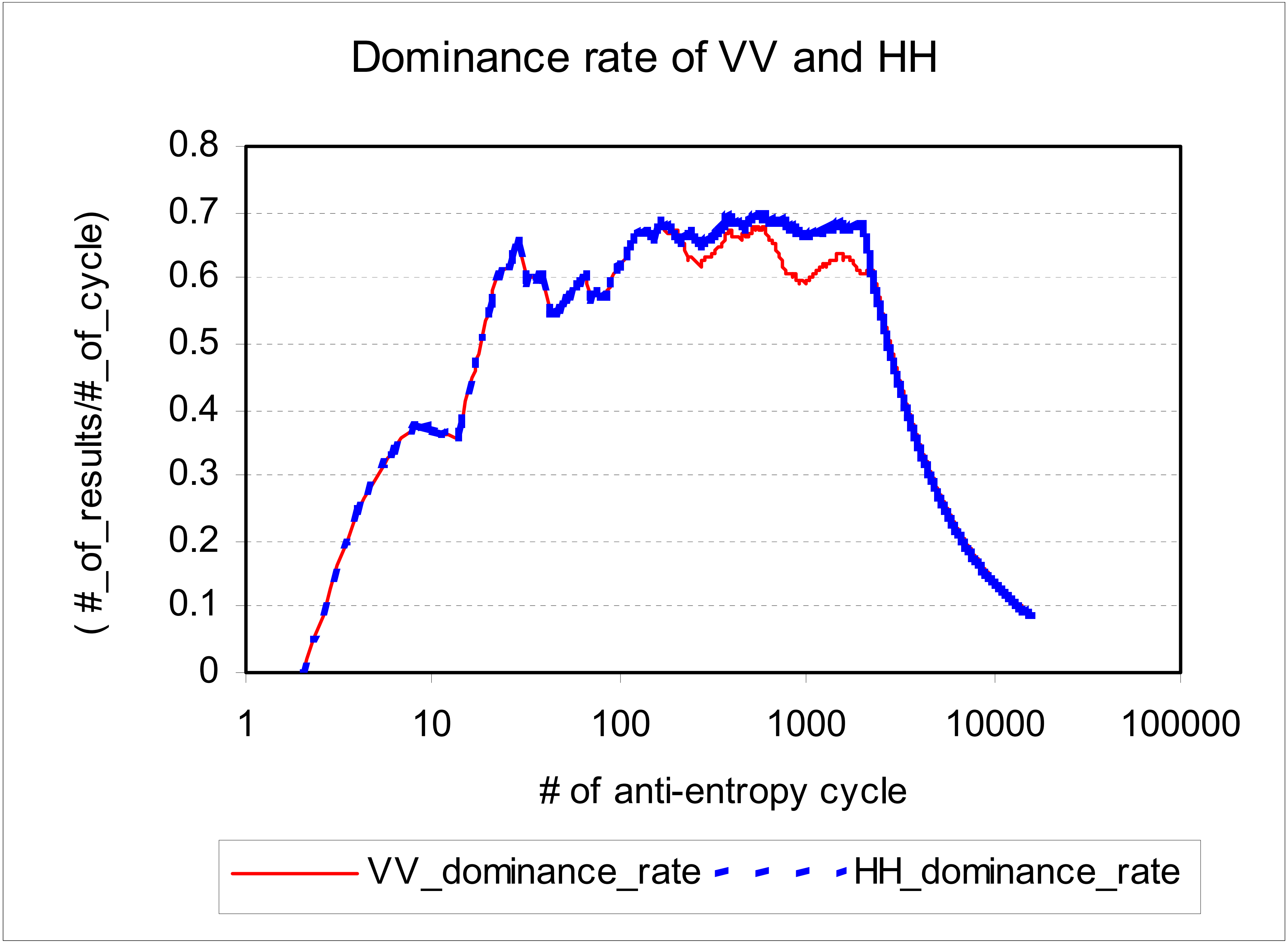
CVS Trace Data (from sourceforge.net)

	Dri	Freenet	Pcgen
# of events	10137	2281	404
# of users	21	64	39
Duration	4/27/1994 - 5/3/2002	12.28.1999 -4/25/2002	1/17/2002 - 4/12/2002
inter-commit time AVG	101.3 min	237.8 min	225.4 min
MEDIAN	0.016 min	34.6 min	2.16 min


Conflict rate of VV and HH
0.6
0.5
0.4
0.3
0.2
0.1
0
(#_of_results/#_of_cycle)
1
10
100
1000
10000
100000
# of anti-entropy cycle
VV_conflict_rate
HH_conflict_rate



Equality rate of VV and HH
(#_of_results/#_of_cycle)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
10
100
1000
10000
100000
# of anti-entropy cycle
VV_equality_rate
HH_equality_rate

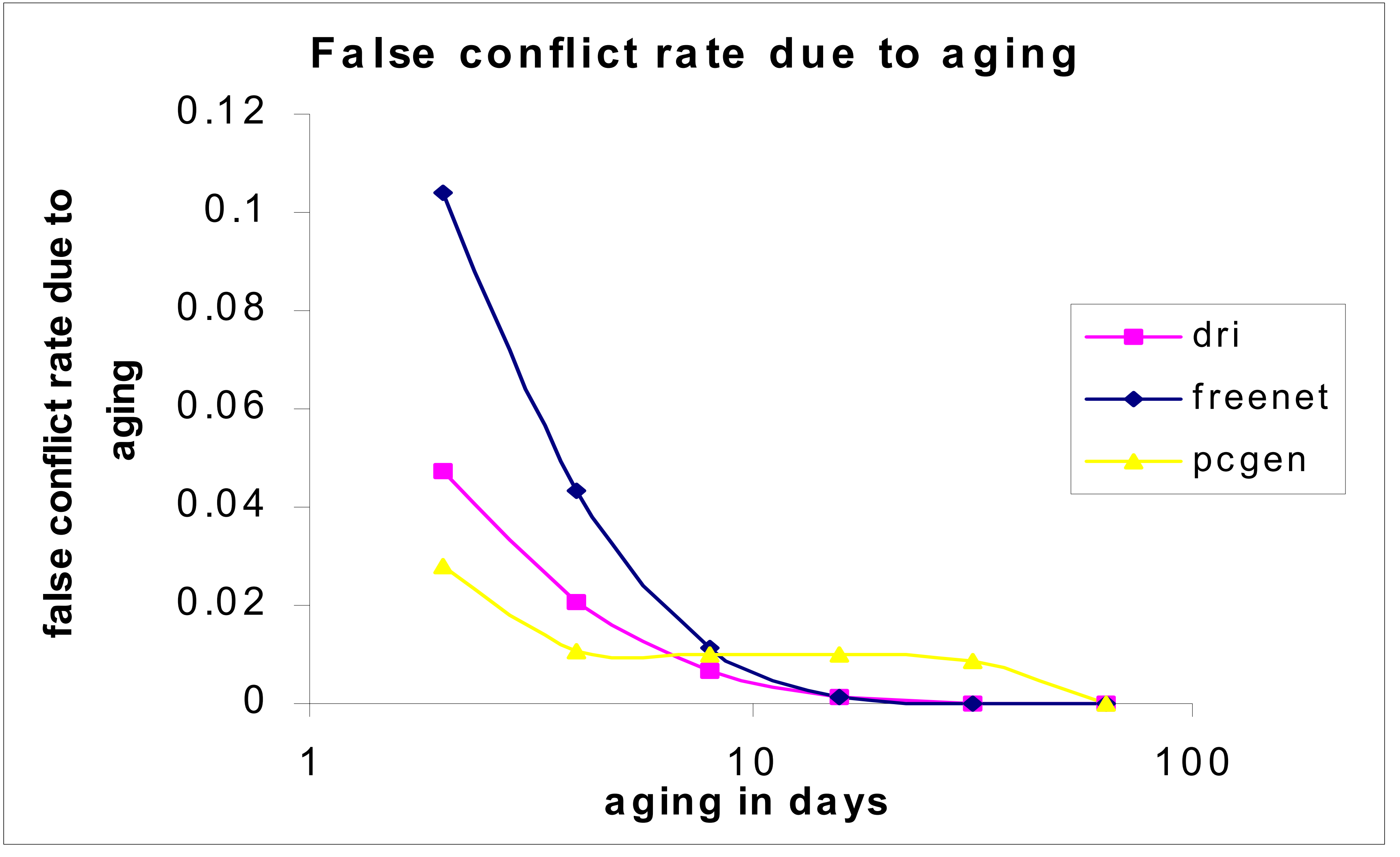


Dominance rate of VV and HH
(#_of_results/#_of_cycle)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1
10
100
1000
10000
100000
# of anti-entropy cycle
VV_dominance_rate
HH_dominance_rate

Aging Period vs. HH Size

Aging period (days)	HH size (# of entries) – dri	pcgen	freenet	Average
32	146.3	159.1	61.5	122.3
64	413.9	443.9	147.5	335.1
128	551.5	591.7	612.8	585.3

Aging Period vs. False Conflict

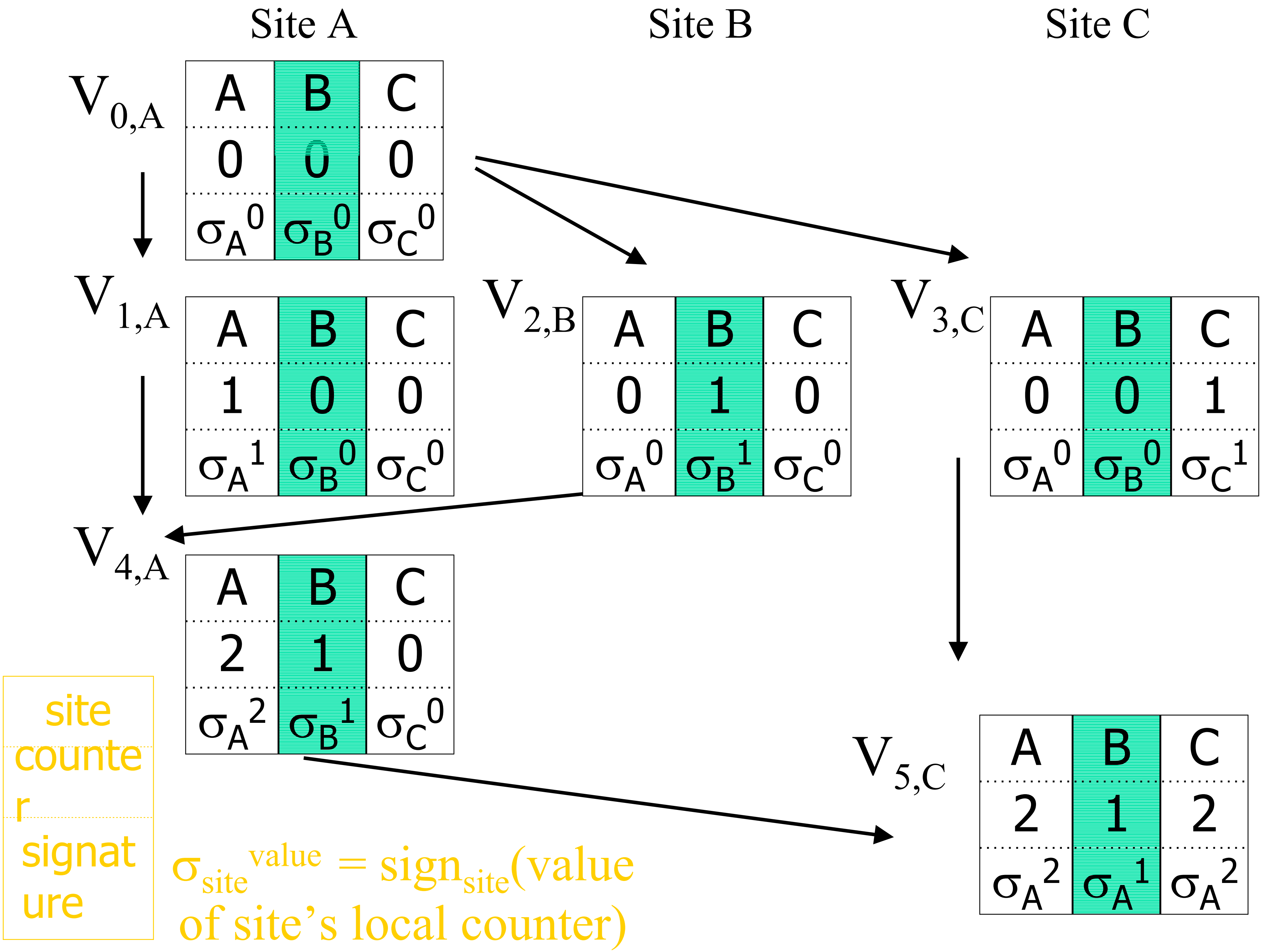


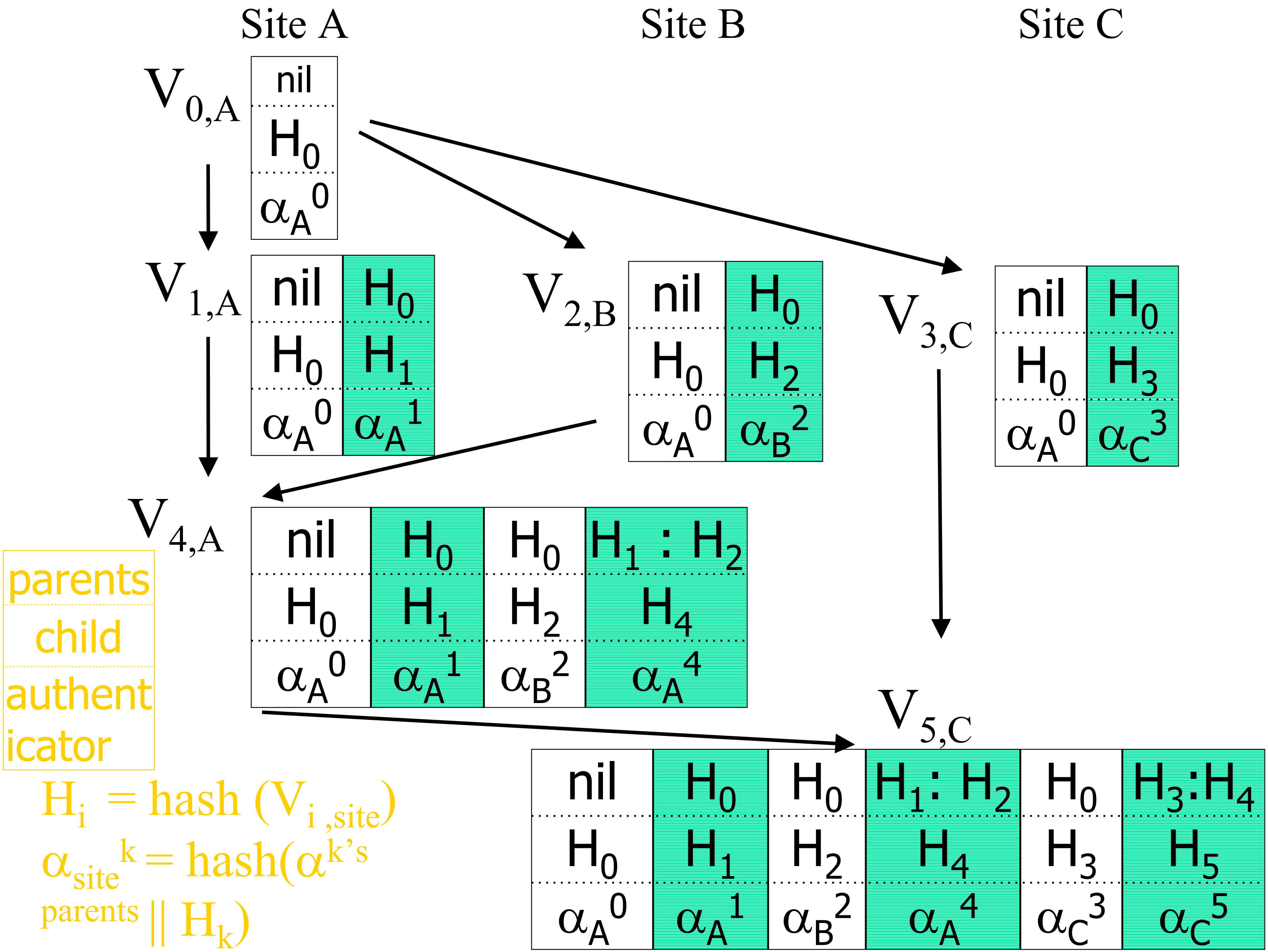
Conclusion

- Simple to maintain
  - No complexity in site addition/deletion
  - No need to assign unique id dynamically for newly added replica sites
- Scalable to thousands of sites
  - HH grows in proportion to number of update instances not number of sites
- Faster Convergence
  - HH can capture and propagate equality information
- HH growth can be controlled effectively by
  - using aging policy or sharing archived hash history

Future Work

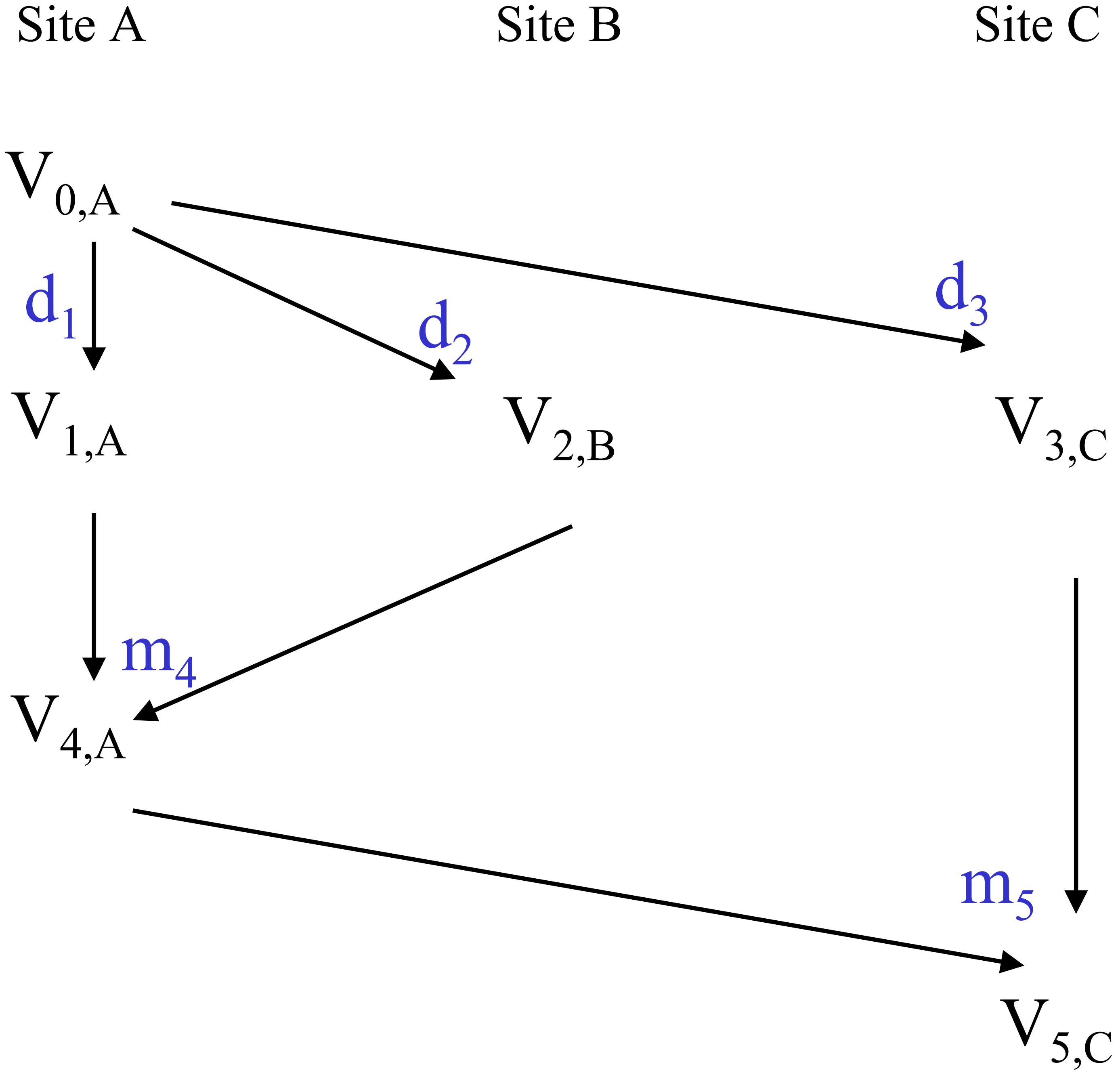
- Security aspect of HH
  - Self-verifiable
  - Can detect mal-functioning site
- More information
  - Hash History Approach for Reconciling Mutual Inconsistency in Optimistic Replication, *B. Kang, R. Wilensky and J. Kubiatowicz*, The 23rd International Conference on Distributed Computing Systems (ICDCS), 2003, Providence, Rhode Island USA
  - http://www.cs.berkeley.edu/~hoon/hashhistory

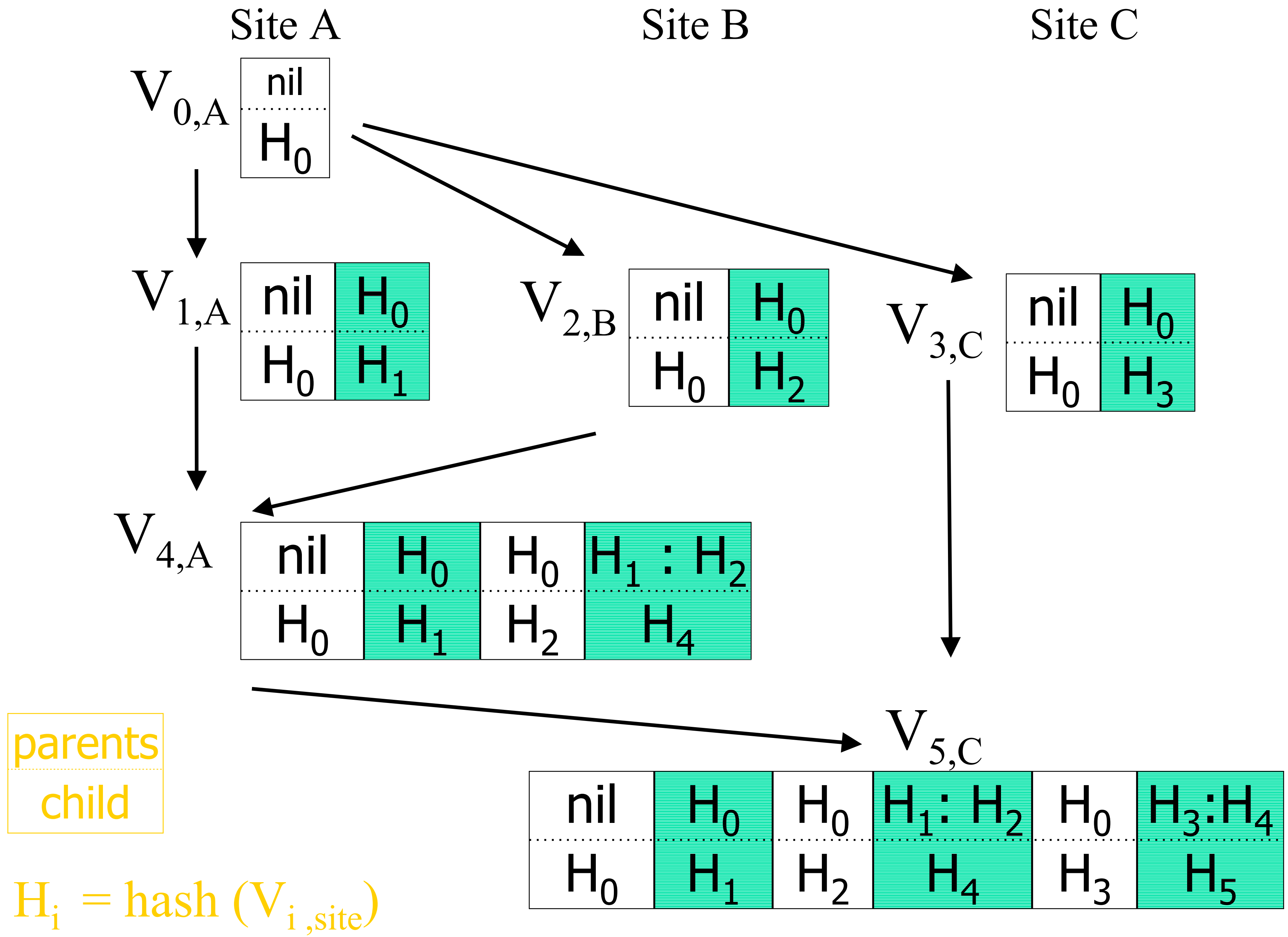

Site A
Site B
Site C
V0,A
A B C
0 0 0
σA0 σB0 σC0
V1,A
A B C
1 0 0
σA1 σB0 σC0
V2,B
A B C
0 1 0
σA0 σB1 σC0
V3,C
A B C
0 0 1
σA0 σB0 σC1
V4,A
A B C
2 1 0
σA2 σB1 σC0
V5,C
A B C
2 1 2
σA2 σA1 σA2
site
counte
r
signat
ure
σsitevalue = signsite(value of site's local counter)



$$H_i = \text{hash}(V_{i,\text{site}})$$

$$\alpha_{\text{site}}^{k} = \text{hash}(\alpha^{k\text{'s parents}} \,||\, H_k)$$


Site A
Site B
Site C
V0,A
d1
d2
d3
V1,A
V2,B
V3,C
m4
V4,A
m5
V5,C



Site A
Site B
Site C
$V_{0,A}$
nil
H_0
$V_{1,A}$
nil
H_0
H_0
H_1
$V_{2,B}$
nil
H_0
H_0
H_2
$V_{3,C}$
nil
H_0
H_0
H_3
$V_{4,A}$
nil
H_0
H_0
H_1
H_0
H_2
$H_1 : H_2$
H_4
$V_{5,C}$
nil
H_0
H_0
H_1
H_0
H_2
$H_1 : H_2$
H_4
H_0
H_3
$H_3 : H_4$
H_5
parents
child
$H_i = hash(V_{i,site})$