# Consistency Models in Distributed Shared Memory: Theory and Practice

Hengfeng Wei

ICS, NJU
hengxin0912@gmail.com

June 5, 2015

# Consistency Models in DSM: Theory and Practice

# Consistency Models in DSM: Theory and Practice

# Distributed shared memory

## Definition (Distributed shared memory (DSM) [Attiya@Wiley'04])

- A model for inter-process communication
- An illusion of shared memory on top of message passing

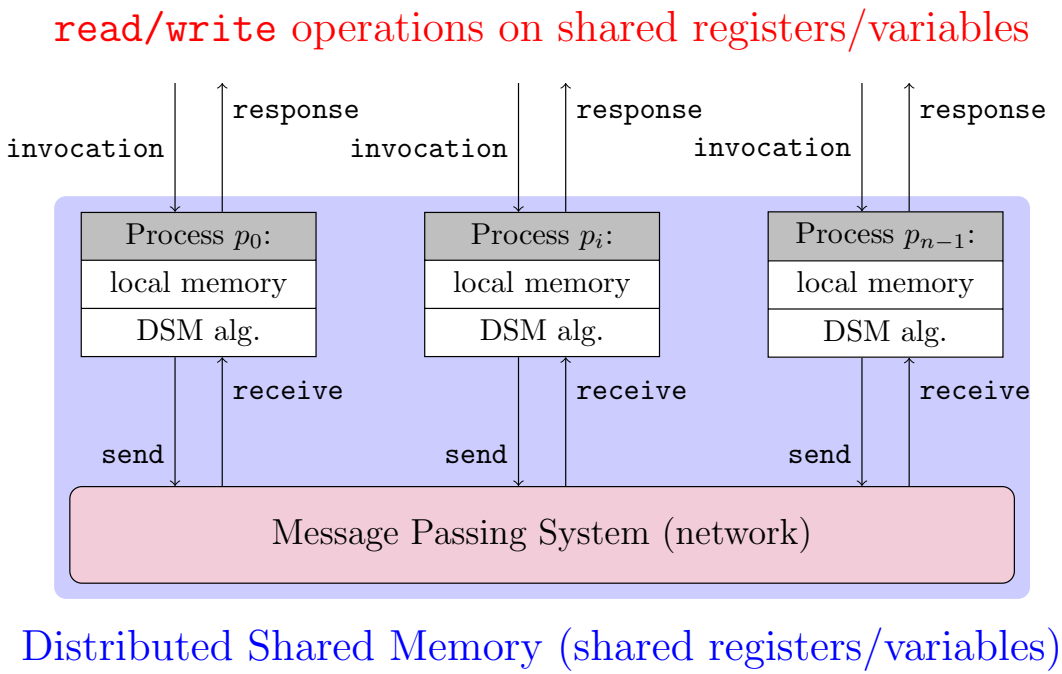



**Figure 1:** A distributed shared memory on top of a message passing system.

# Benefits of DSM

**A more global view of asynchronous distributed system:**

- familiar programming model for programmer
  - `read/write` vs. `send/receive`
- simple computational model for theoretician
  - e.g., distributed recursion

# DSM in practice


**Figure 2:** Distributed storage systems (open source [left] & commercial [right]).

# DSM in practice

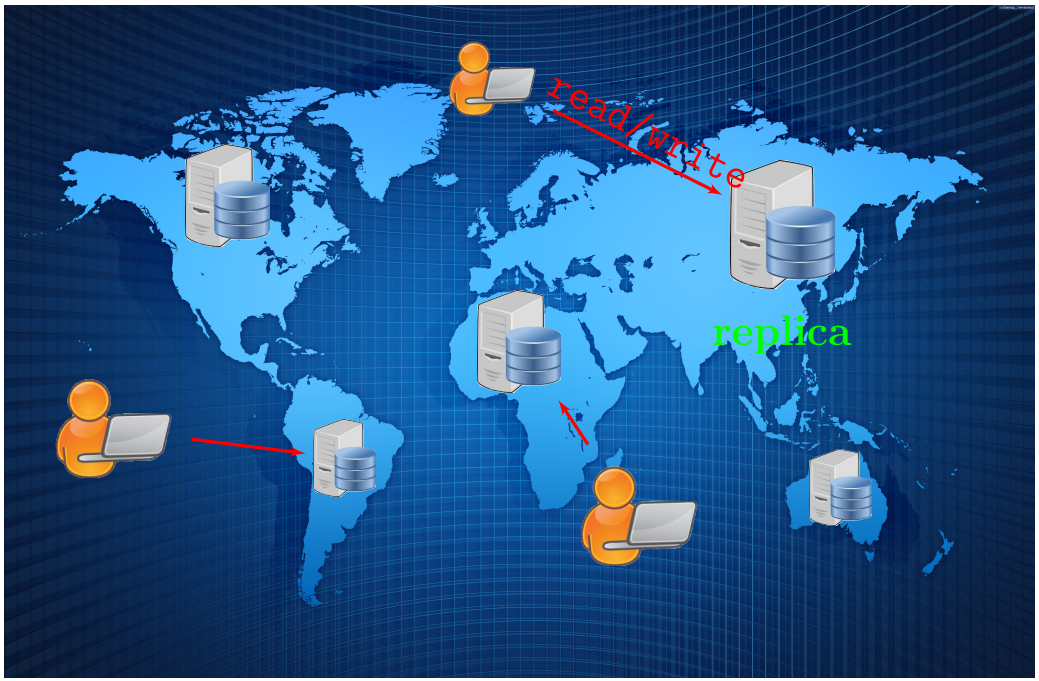



**Figure 3:** Architecture (simplified) of distributed storage systems.

# Distributed shared memory issues

**When a process `invokes` a dsm operation (`read`/`write`):**

1. executes local computations
2. exchanges messages with others
3. `responds` to the process

**DSM operations are *NOT* instantaneous!**

**No problem if there is only one process:**

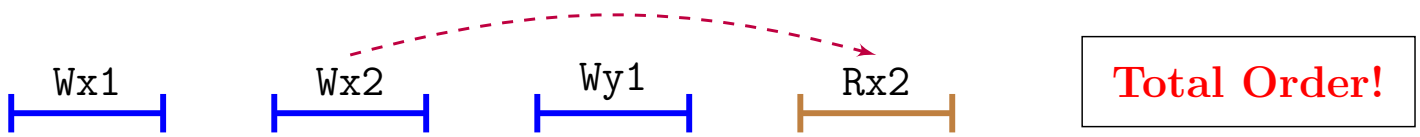
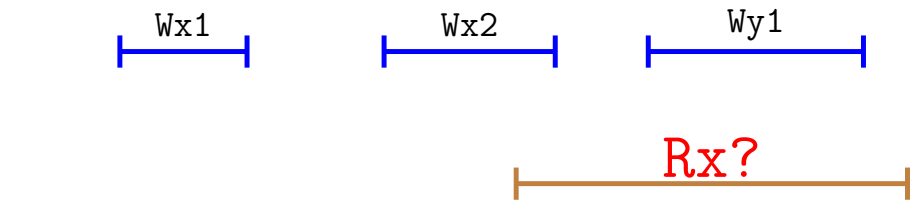



**Figure 4:** Sequential `read` semantics: `read` reads the latest `write`.

# Distributed shared memory issues

**When multiple processes `invoke` operations concurrently:**




NO Total Order!

**The Question:**

What value(s) should be returned by a `read` that overlaps `writes`?

**The Answer:**

Defined by memory consistency models.

# Consistency models of DSM

**Definition (Consistency model [Steinke@JACM'04])**

A consistency model for dsm:

- specifies the values may be returned by *reads*
  - even though operations are only partially ordered

**that is, specification of the allowable behavior of dsm.**

**thereby defining the semantics of shared registers/variables.**

# Benefits of consistency models

**A *Consistency Model* is a contract between dsm and users**

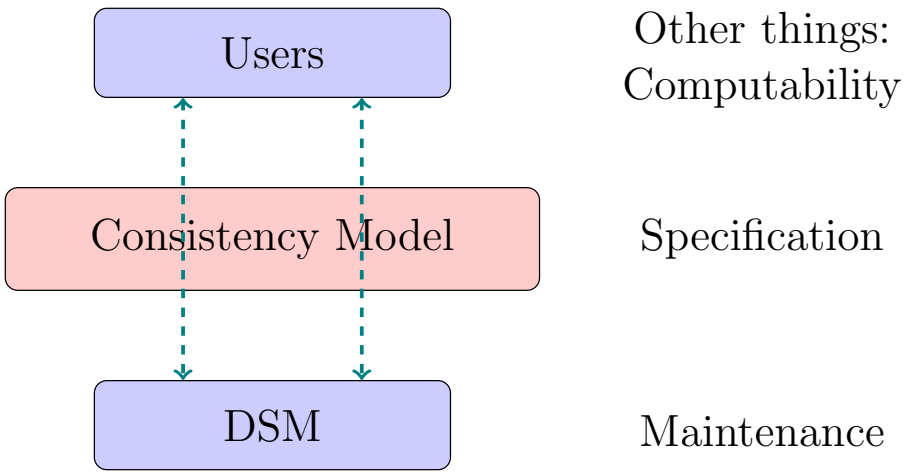



**Figure 5:** Consistency model between dsm and users.

# Consistency Models in DSM: Theory and Practice

# Consistency models in DSM

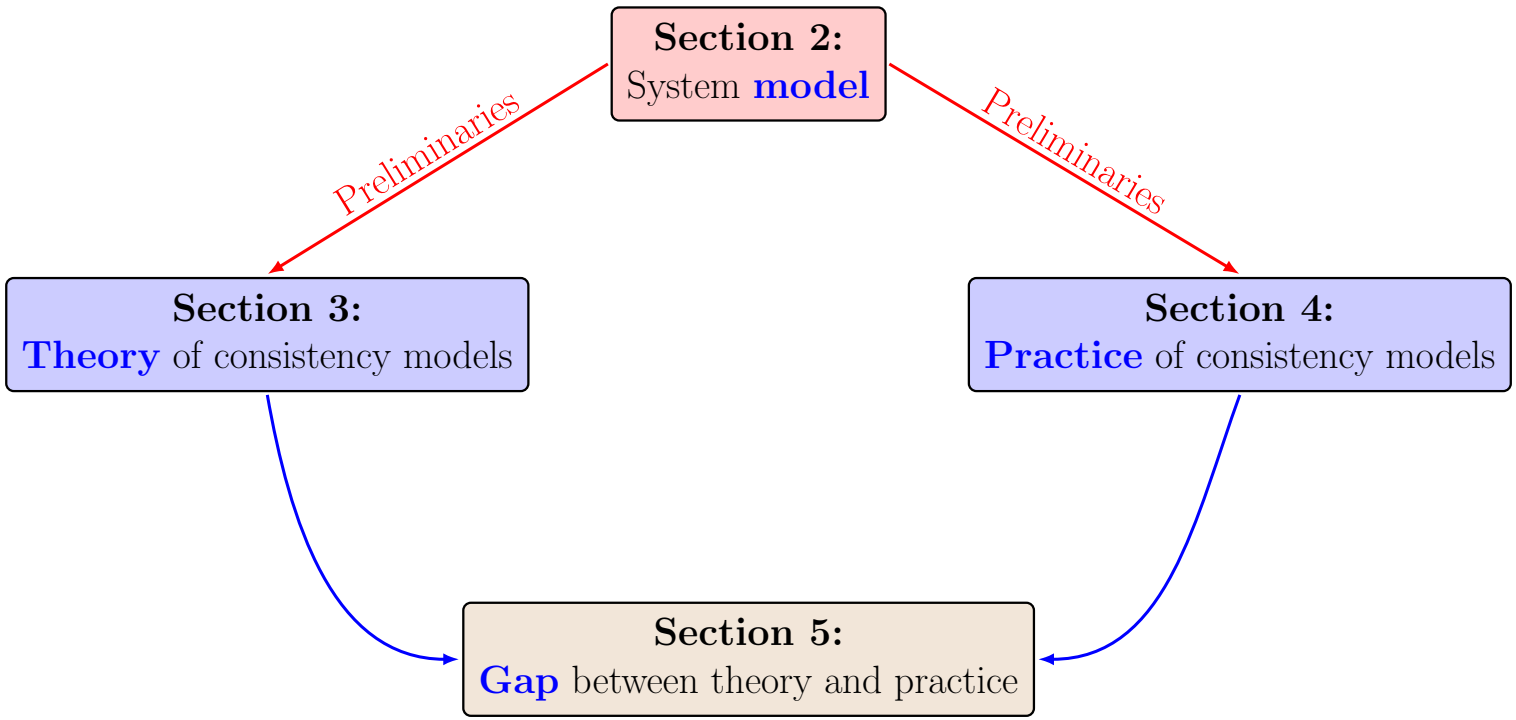



**Figure 6:** Contents of the survey on consistency models in dsm.

# Theory of consistency models (Section 3)

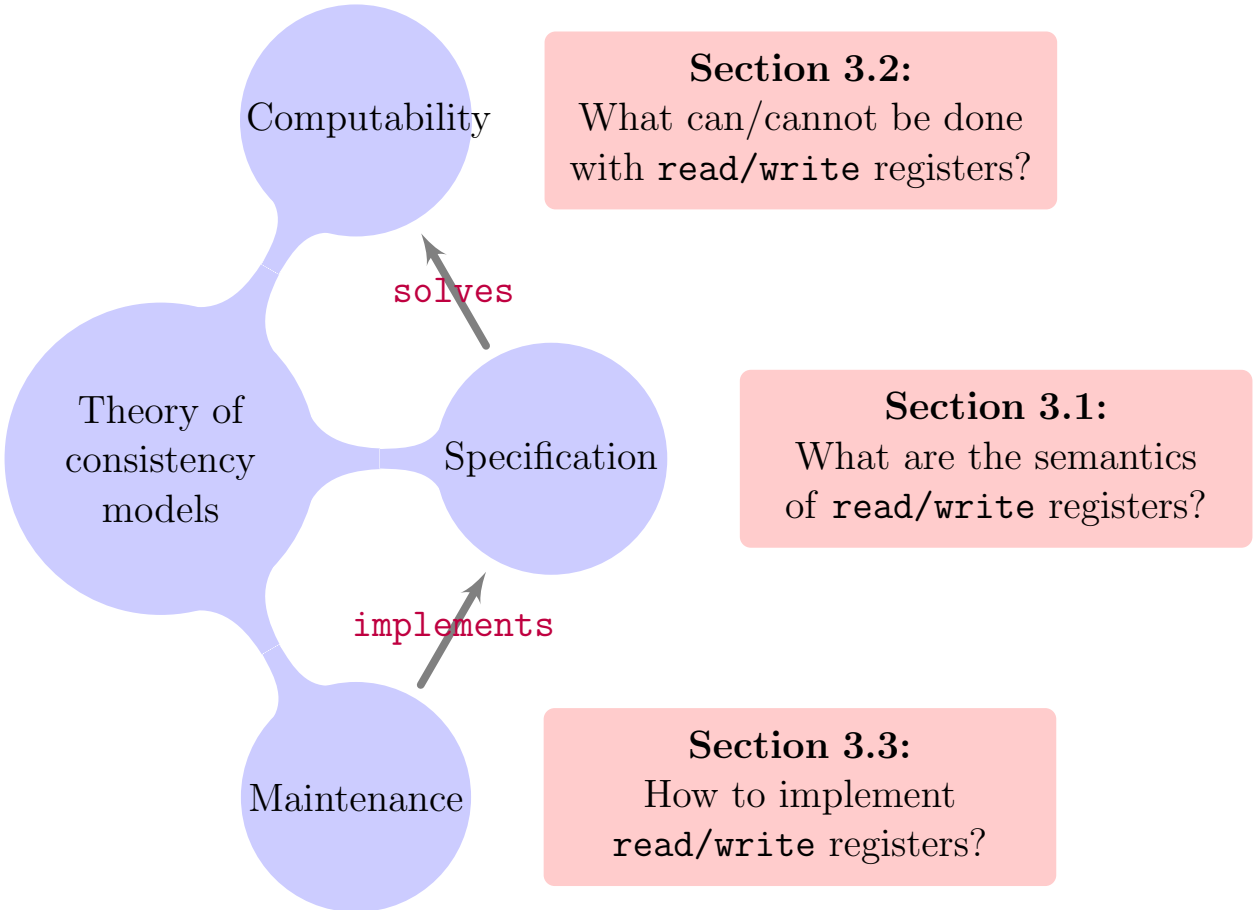



**Figure 7:** Specification, computability, and maintenance in the theory of consistency models.

# Practice of consistency models (Section 4)

**Practice of consistency models in distributed storage systems (both commercial and open-source): Trade-offs everywhere,**

1. trade-offs from the perspective of consistency
2. design elements from the perspective of consistency

| Consistency models | Design elements | | |
|---|---|---|---|
| | **Topology** | **Concurrency control** | **Ordering policy** |
| Weak consistency models | | | |
| Strong consistency models | | | |
| Hybrid consistency models [1] | | | |

**Table 1:** Explore design elements from the perspective of weak, strong, and hybrid consistency models in distributed storage systems.

[1] Go to Backup Slides for Hybrid Consistency Models

# Gap between theory and practice (Section 5)

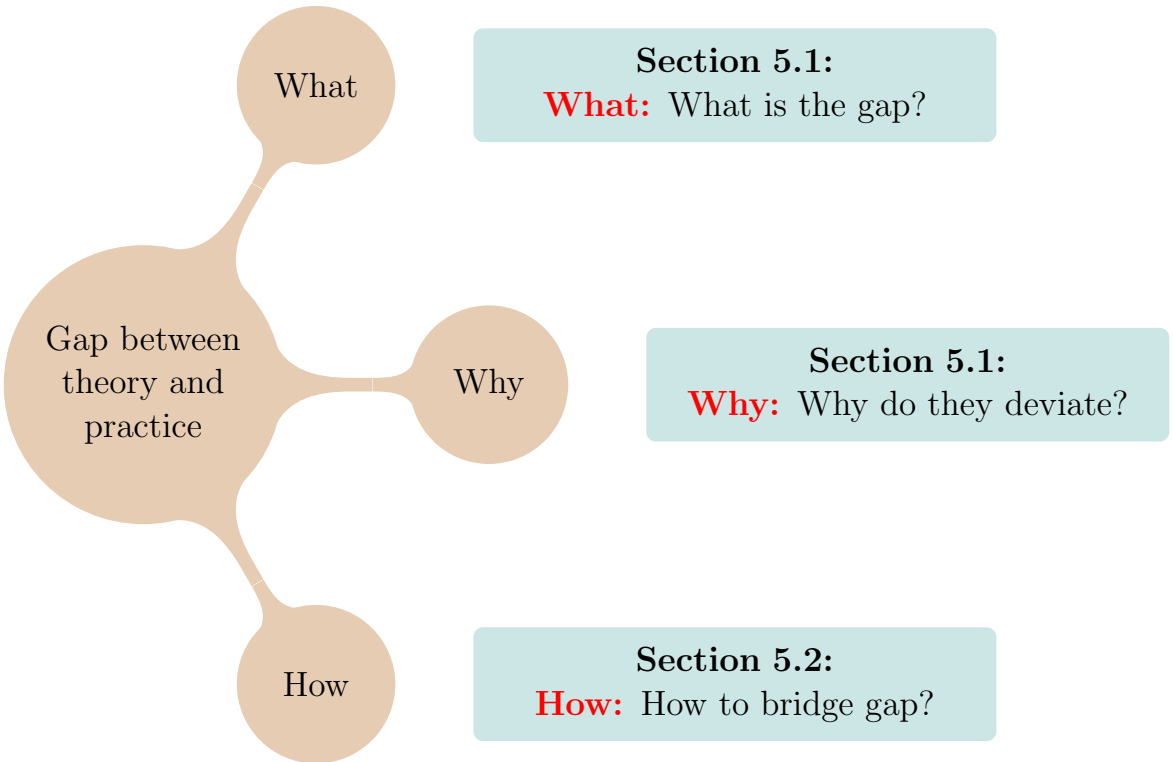



**Figure 8:** Gap between theory and practice of consistency models.

# In scope

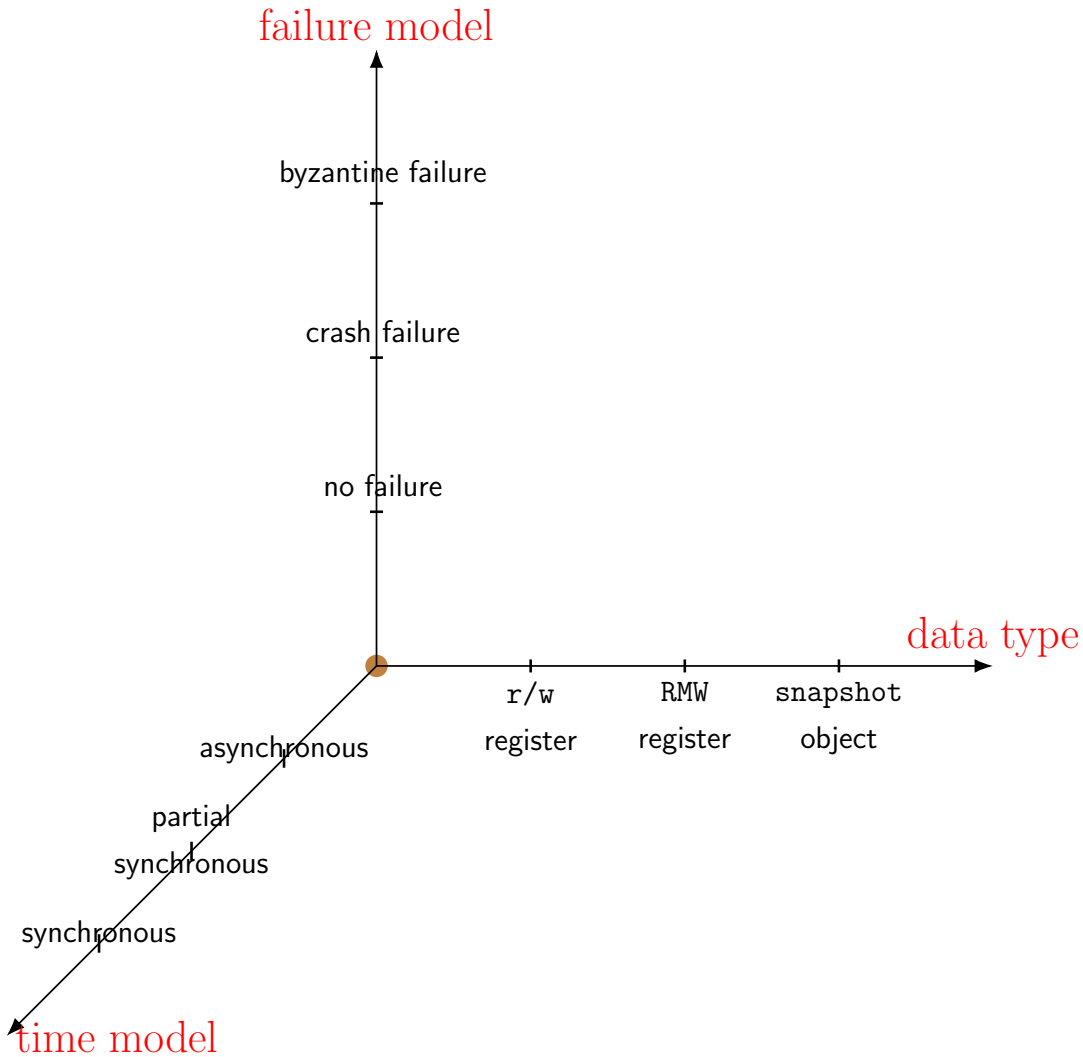



**Figure 9:** The three-dimensional space of possible perspectives on consistency models.

# In scope

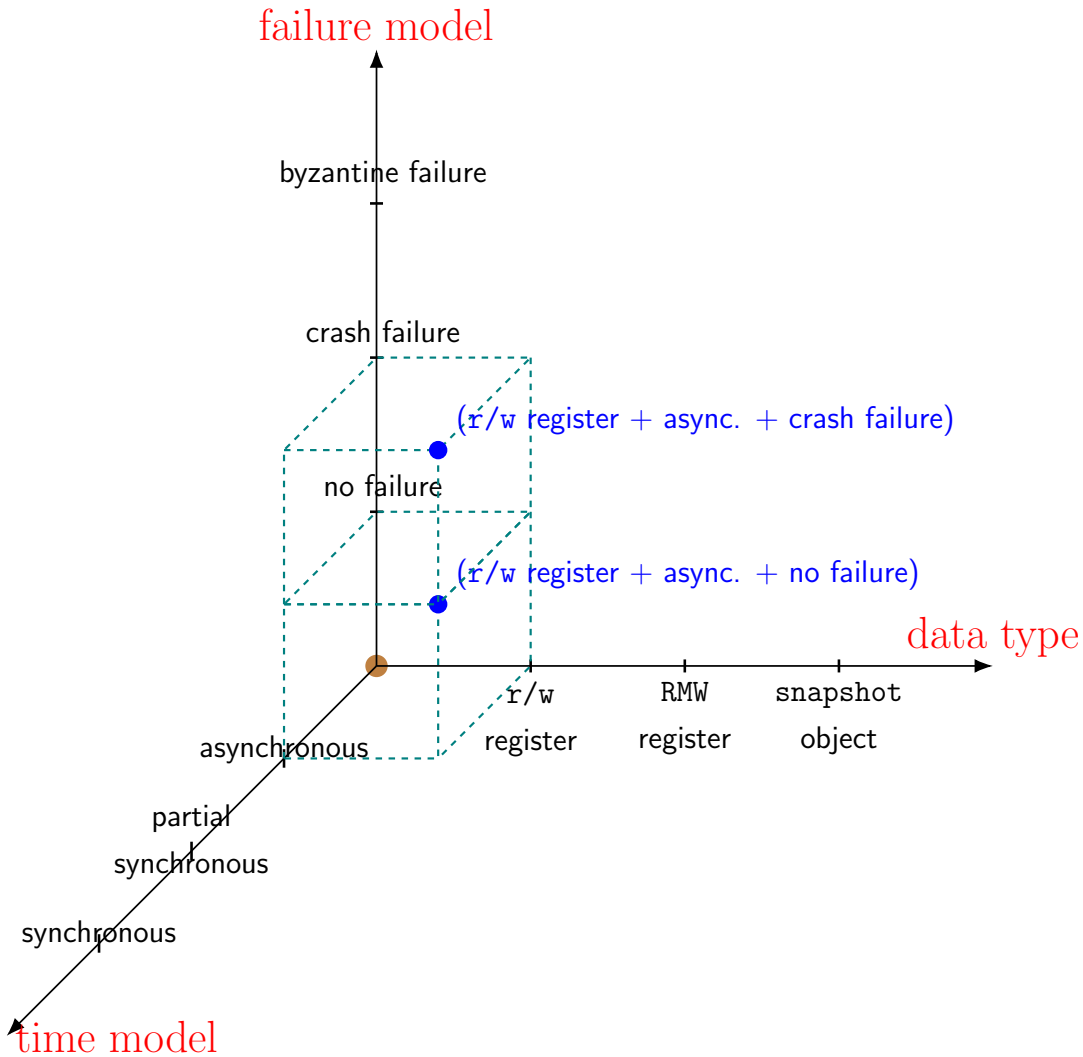



**Figure 9:** The three-dimensional space of possible perspectives on consistency models.

# Out of scope

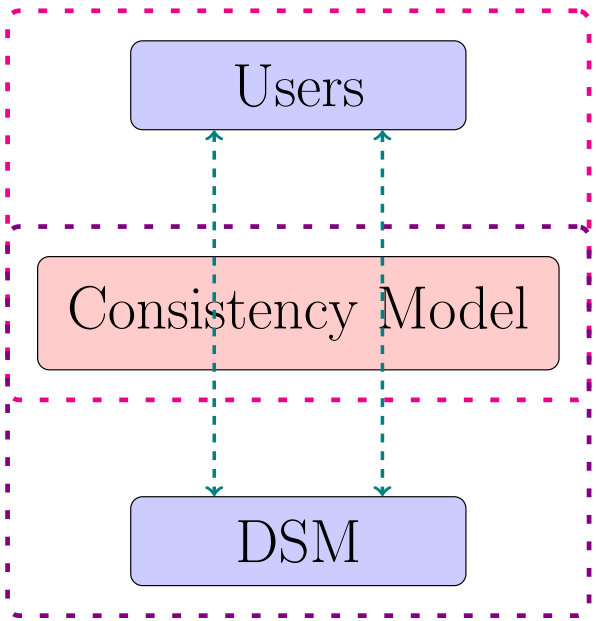
Consistency models in "shared memory multiprocessors" which may be subject to [Adve@IEEE Computer'96], [Adve@CACM'10]

- computer architectures (`e.g., write buffer`)
- compiler optimizations (`e.g., reordering`)
- programming languages (Java™ [Manson@POPL'05], C++ [Boehm@PODI'08])

**The principles are similar, but the details differ a lot !**

# Consistency Models in DSM: Theory and Practice

# Two layers system model




**The specification layer:**
How do users interact with dsm?

**The maintenance layer:**
How is dsm implemented
on top of message passing?

**Figure 10:** Two layers system model: specification and maintenance.

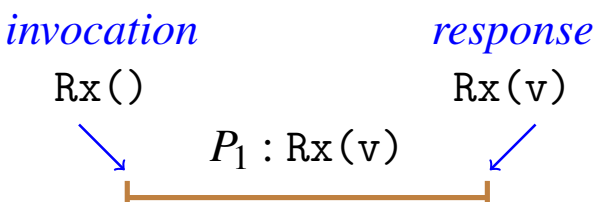
## System model at the specification layer

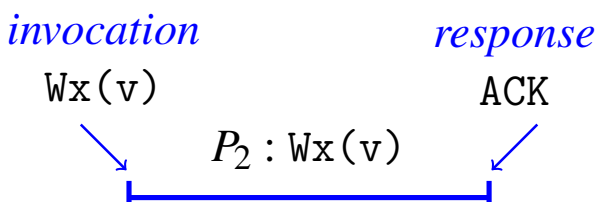
- a set of processes: $\mathcal{P} = \{P_1, P_2, \cdots, P_n\}$
- a set of (distributed) shared registers: $\mathcal{X} = \{x, y, z, \cdots\}$
- operations $\mathcal{O} = \{\texttt{read(x)}, \texttt{write(x,v)}\}$

**Processes communicate via accessing the shared registers.**

# Operations

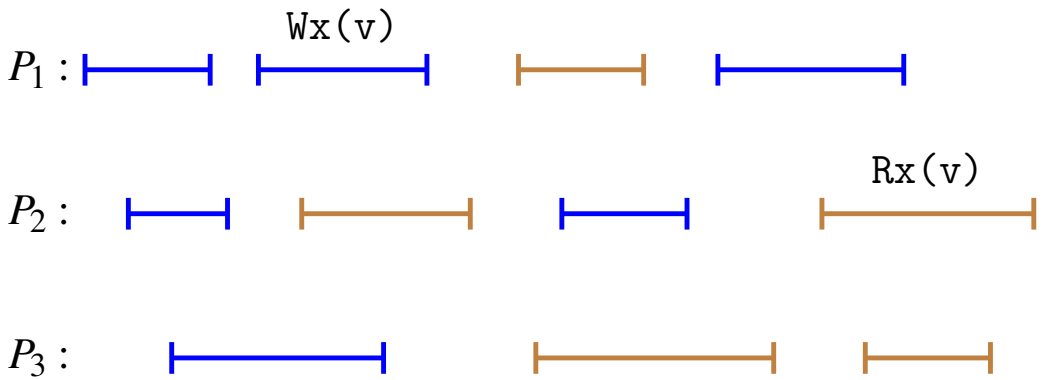
An operation is an interval indicated by two events:
an `invocation` followed by a `response`.



# Executions

An execution is a collection of operation sequences, one per process.



# Time in executions

## Another view of execution

An execution is a sequence of events, in chronological order according to "real-time".

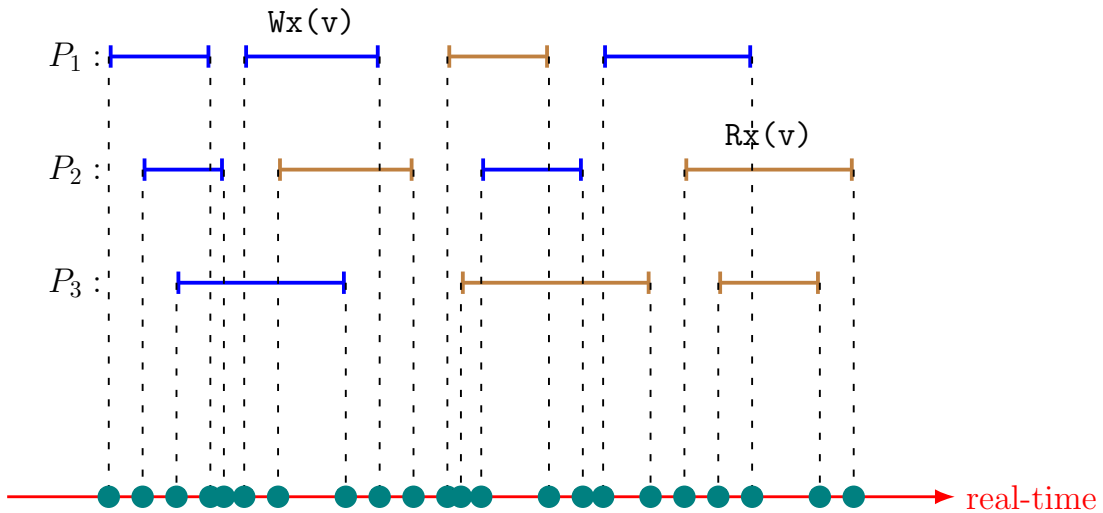



**Figure 11:** Mapping all events in an execution to "real-time" line.

**The "real-time" exists but is not available for maintenance.**

# System model at the maintenance layer

**For maintenance, we assume an asynchronous message passing system:**

- no global clock
- no bound on message delay
- no bound on the speed of a process

**Two uncertainties:**

- cannot tell whether a message is delayed or lost
- cannot tell whether a process is slow or failed

# Consistency Models in DSM: Theory and Practice

# Consistency Models in DSM: Theory and Practice

# Specification of consistency models

**The Basic Question:**

What value(s) may be returned by a `read` in an execution?

**Challenge:** processes interleave $\Rightarrow$ partial order of operations

$$\textbf{different specifications} \underset{\textbf{define}}{\overset{\textbf{specify}}{\rightleftharpoons}} \textbf{different partial orders}$$

# Specification of consistency models

**The Basic Question:**

What value(s) may be returned by a `read` in an execution?

**Challenge:** processes interleave $\Rightarrow$ partial order of operations

$$\textbf{different specifications} \underset{\textbf{define}}{\overset{\textbf{specify}}{\rightleftharpoons}} \textbf{different partial orders}$$

**Whether an execution conforms to some specification:**

It should *behave like* a sequential execution:

1. sequential: total order + `read` semantics
2. partial order $\xRightarrow{\texttt{linearized}}$ multiple total orders
3. $\exists$ a total order: `read` reads from the latest `write`

# Real-time order

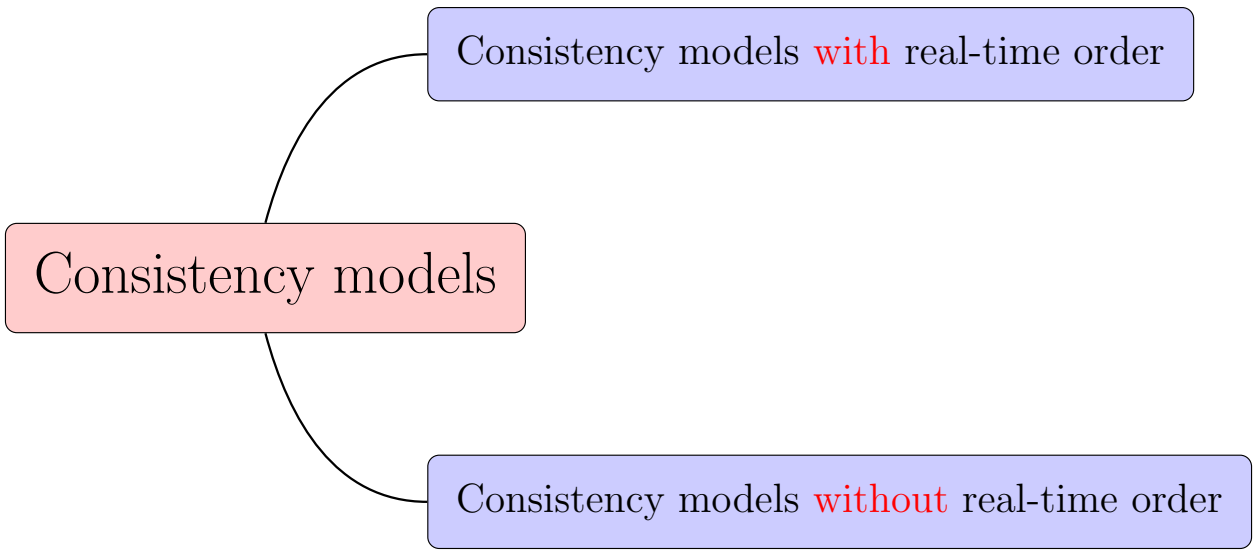
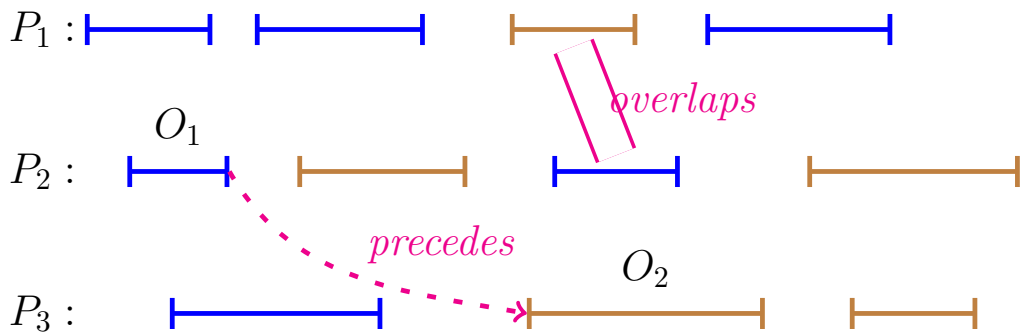



**Figure 12:** Classifying consistency models by respecting the *real-time order* or not.

# Real-time order




**Definition (Precedes relation '$\prec$' and overlaps relation '$\|$')**

$o_1 \prec o_2 \iff o_1.\text{response} <_t o_2.\text{invocation}$

$o_1 \parallel o_2 \iff \neg(o_1 \prec o_2 \vee o_2 \prec o_1)$

**Definition (Real-time order)**

Precedence defines a partial order — the real-time order.

# Consistency models with real-time order

| Consistency models | Value for `read` [2] | The reference |
|---|---|---|
| Atomicity [3] | latest | [Lamport@JACM'86] |
| Regularity (`single-writer`) | latest $\vee$ concurrent | [Lamport@DC'86] |
| Safe (`single-writer`) | arbitrary | [Lamport@DC'86] |
| Linearizability [4] | latest | [Herlihy@TOPLAS'90] |
| Quiescent consistency | don't care | [Shavit@TOCS'96] |
| Regularity (`multi-writer`) | it depends [5] | [Shao@DISC'03] |
| Safe (`multi-writer`) | to be defined | [Future Work@?' ?] |

**Table 2:** A chronological list of consistency models *with* real-time order.

[2] We can focus on only the `reads` overlapping `writes`.

[3] Go to Backup Slides for Atomicity as An Example

[4] Linearizability is a generalization of atomicity on arbitrary objects.

[5] There are too many variants [Shao@DISC'03].

# Consistency models without real-time order

| Consistency models | Key "order" | The reference |
|---|---|---|
| Sequential consistency [6] | global sequential order | [Lamport@TC'79] |
| PRAM consistency | global program order (gpo) | [Lipton@TR'88] |
| Processor consistency | gpo + hf | [Goodman@TR'89] |
| Cache consistency | sequential on each register | [Goodman@TR'89] |
| Causal consistency | "happened-before" (hb) | [Ahamad@WDAG'91] |
| Eventual consistency | no order | [Terry@SOSP'95] |
| Local consistency | local program order | [Bataller@Euro-Par'97] |

**Table 3:** A chronological list of consistency models *without* real-time order.

[6] Go to Backup Slides for Sequential Consistency as An Example

# Frameworks for consistency models

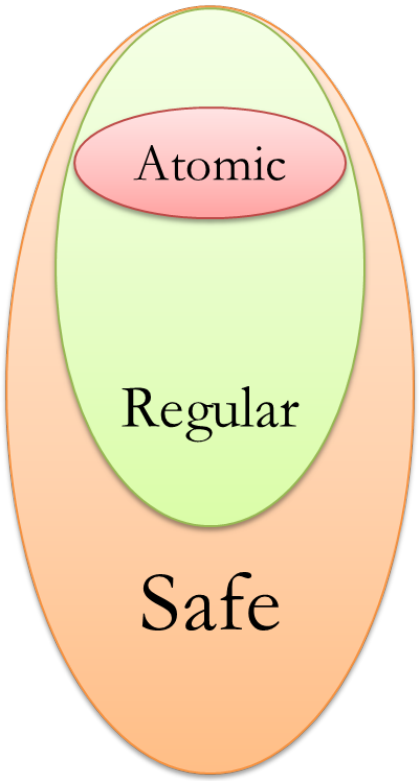



**Figure 13:** Hierarchy of consistency models with real-time order [Lamport@DC'86], [Haldar@JACM'07].

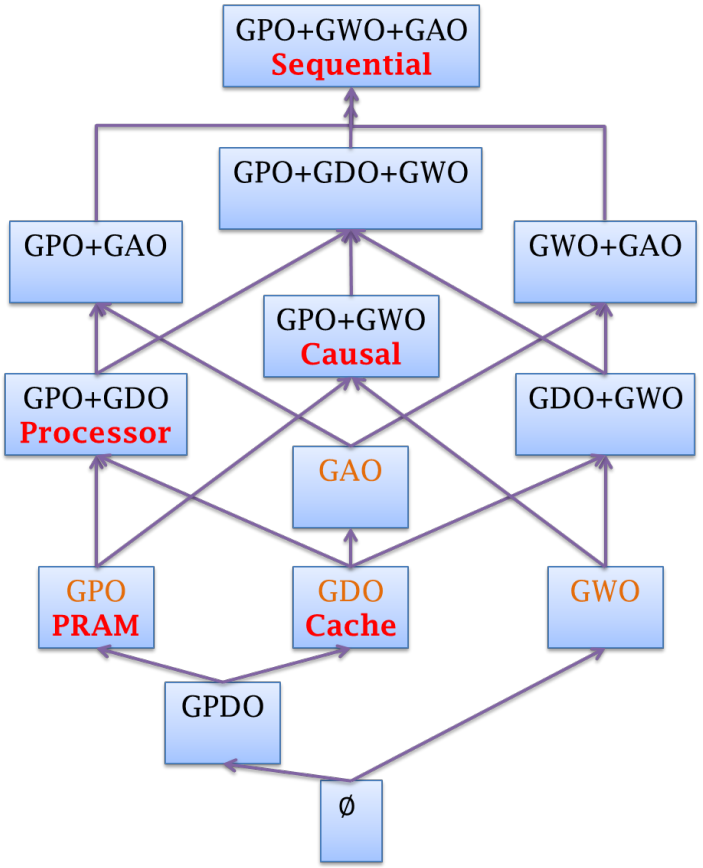



**Figure 14:** A unified theory of shared memory consistency: the lattice of consistency models without real-time order [Steinke@JACM'04].

# Consistency Models in DSM: Theory and Practice

# Computability: how powerful are registers?

**In this report, we take the `Read/write` register semantics with real-time order as example:**

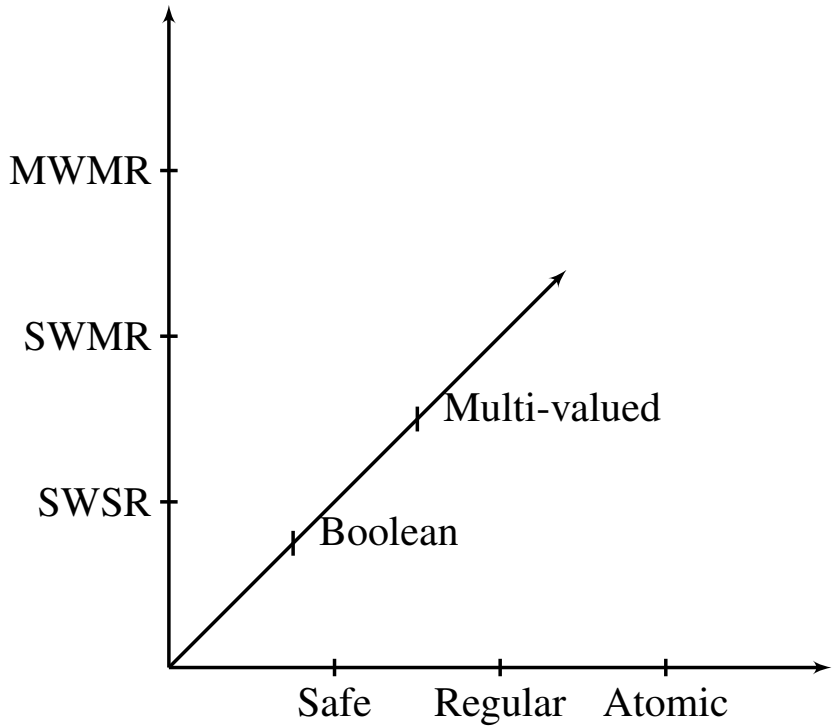



**Figure 15:** The three-dimensional space of `read/write` registers.

# Relative computability powers of registers

**Theorem**

*All types of registers are computationally equivalent.*

**Proof.**

By simulation.

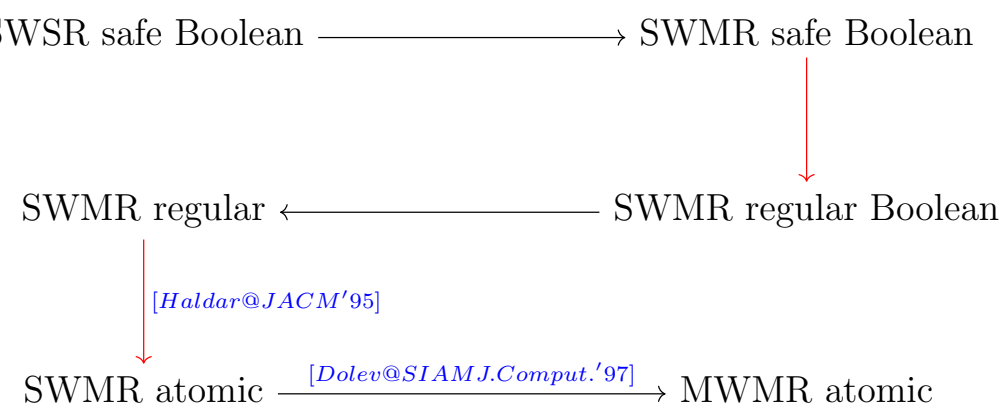



**Figure 16:** Simulations between `read`/`write` registers — A big fad around 1990s; still under research.

# Computability limitations of registers

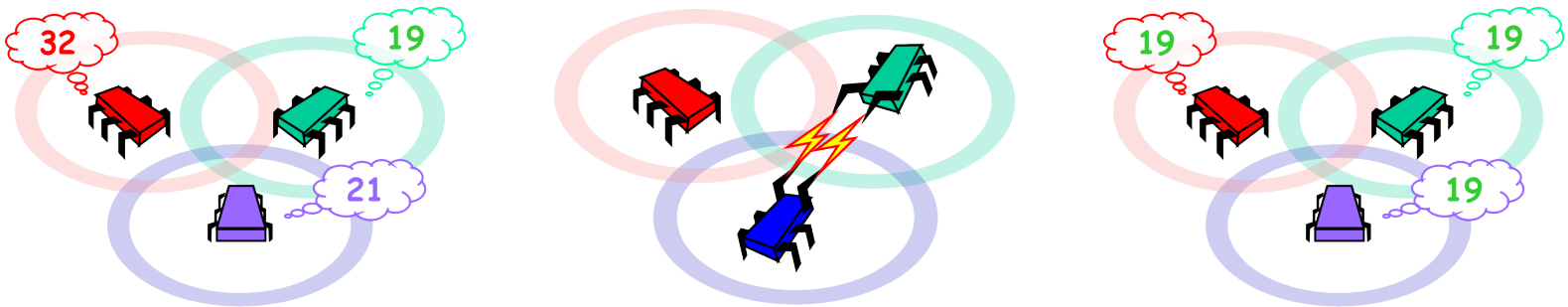



**Figure 17:** The consensus problem illustrated.

# Computability limitations of registers

| Consensus numbers | Concurrent objects |
|---|---|
| 1 | `read/write` registers [FLP@JACM'85], [Herlihy@TOPLAS'91] |
| 2 | get&set, swap, queue, stack |
| ⋮ | ⋮ |
| $2n-2$ | $n$-register assignment |
| ⋮ | ⋮ |
| $\infty$ | compare&swap, augmented queue |

**Table 4:** Consensus numbers and consensus hierarchy [Herlihy@TOPLAS'91].

# Computability powers of registers

**Theorem**

*The weakest read/write registers — the SWMR safe Boolean* [7] *ones — are powerful enough to solve the mutual exclusion problem.*

**Proof.**

By the bakery algorithm [Lamport@CACM'74]. □

`Read/write` **registers are also powerful enough to solve:**

- ▷ atomic snapshot [AADGMS@JACM'93]
- ▷ immediate atomic snapshot [Borowsky@PODC'93]
- ▷ renaming [Attiya@Wiley'04]

[7] It is circular (and wrong) to solve the mutual exclusion problem assuming atomic `read/write` registers [Lamport@alllamportspubsontheweb'15].

# Consistency Models in DSM: Theory and Practice

# System model at the maintenance layer reviewed

## Maintenance:

- How to implement a shared `read/write` register on top of message passing?
- What happens between $[o_{\texttt{invocation}}, o_{\texttt{response}}]$?

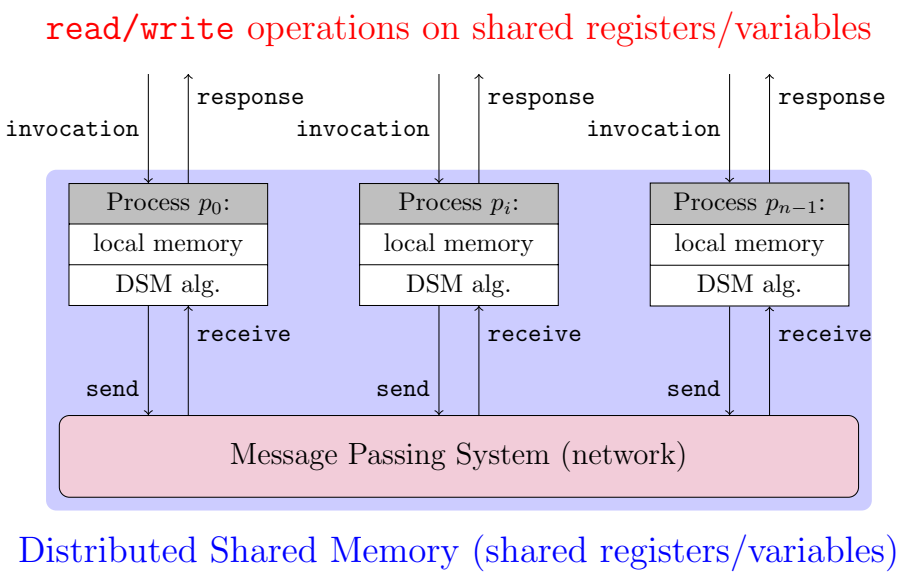


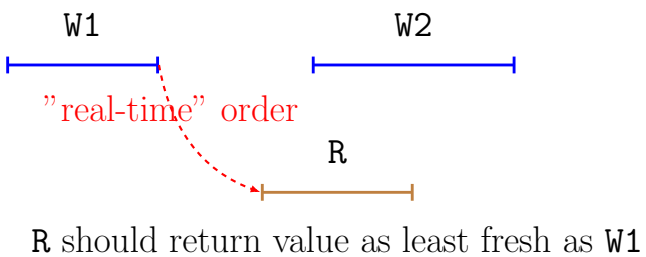

**Figure 18:** A dsm on top of message passing reviewed.

# General techniques for maintenance

To establish operation orders without using "real-time":

1. quorum systems for real-time order
2. timestamps for other orders

## Quorum systems for establishing real-time order: basic idea



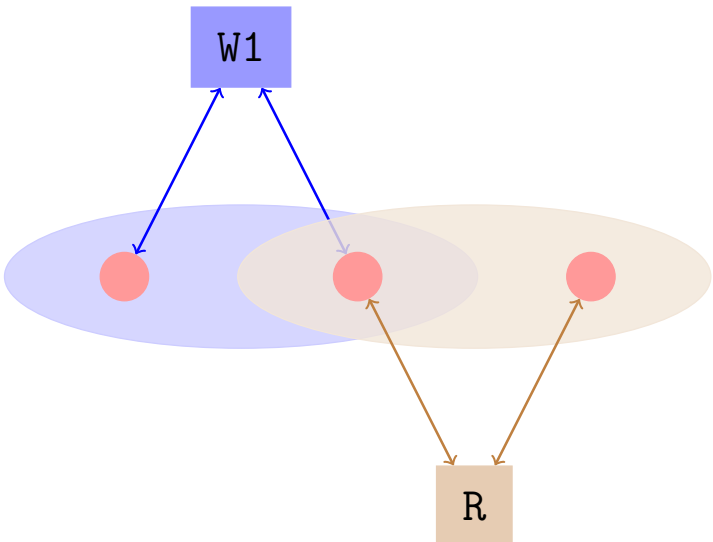




**Figure 19:** Both W1 and R access a majority of (2 out of 3) replicas before completing.

# Timestamps for maintenance

**Timestamps for establishing other orders:**

1. program (FIFO) order: *local counter*
2. causal order: *vector clock* [Lamport@CACM'78], [Fidge@ACSC'88]
3. sequential (total) order: *(local counter, pid)*

# Maintenance

**We analyze the maintenance algorithms (for different consistency models) in terms of quorum systems and timestamps.**

Go to Backup Slides for Maintenance Algorithms

# Consistency Models in DSM: Theory and Practice

# Consistency Models in DSM: Theory and Practice

# Distributed storage systems

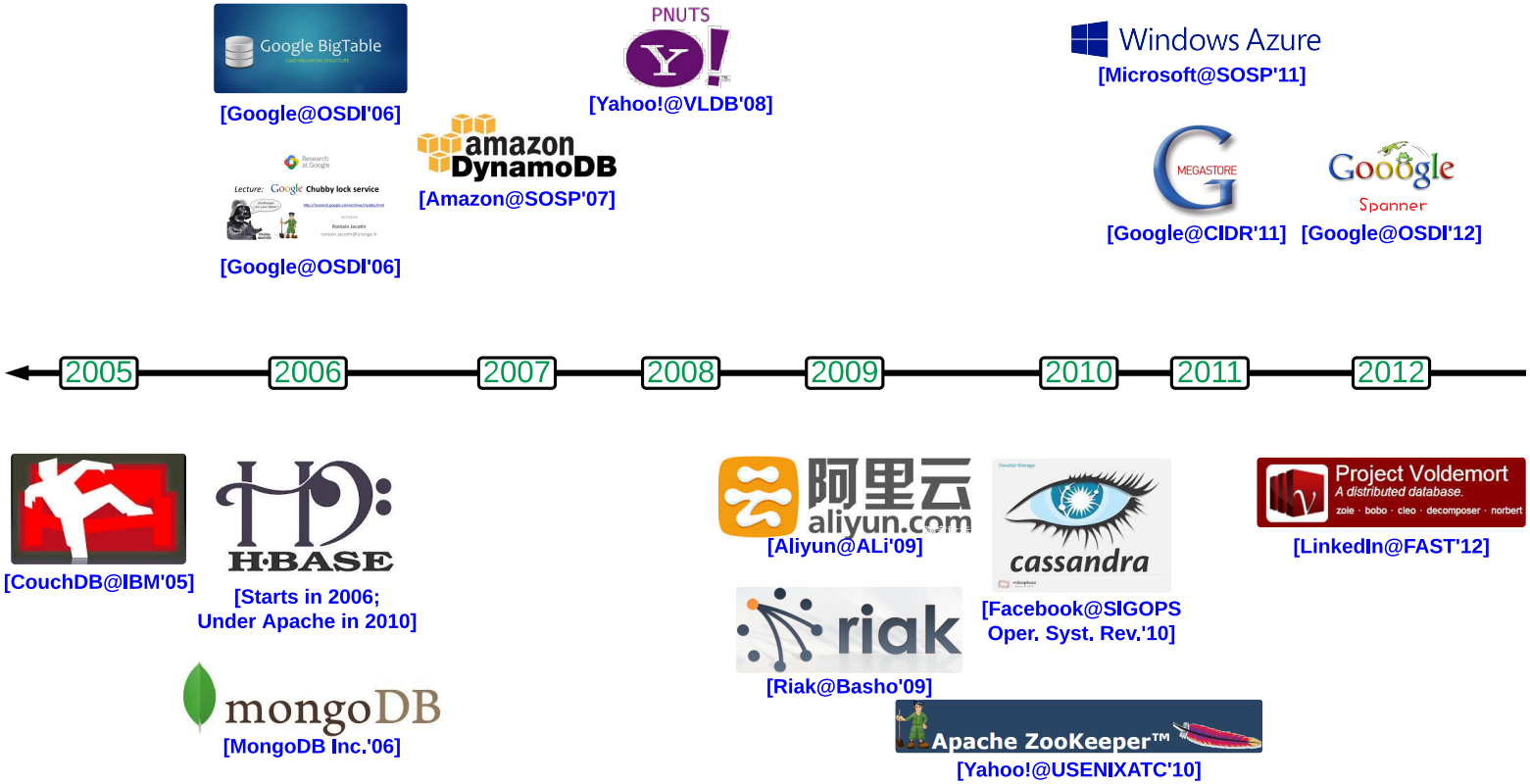


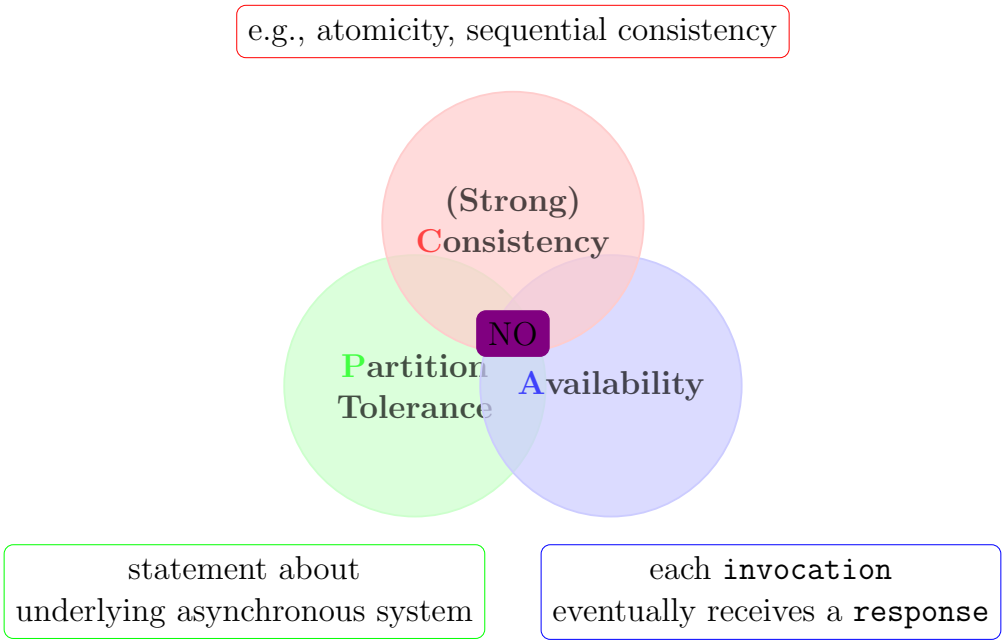

**Figure 20:** A chronological list of distributed storage systems (commercial [above] and open-source [below]).

# Desired properties of distributed storage systems

**Desired properties:**

- Consistency
- Availability
- Partition-tolerance
- Latency (performance)
- Reliability (fault-tolerance)
- Scalibility

# The CAP trade-off




**Theorem (The CAP theorem [Brewer@PODC'00], [Gilbert@IEEE Computer'12])**

*It is impossible for any distributed shared-data system to provide C, A, and P simultaneously.*

# The PACELC [pronounced "pass-elk"] trade-off

**Definition (The PACELC trade-off [Abadi@IEEE Computer'12])**

If Partition

Availability &
Consistency

Else

Latency &
Consistency

# Trade-offs from the perspective of consistency

Consistency options:

1. weak consistency models
   - ARGUE: strong consistency is too expensive
2. strong consistency models
   - ARGUE: weak consistency is hard to understand
3. hybrid consistency models

**Definition (Strong Consistency)**

In a strong consistency condition, all processes should agree on the same view of the order in which operations occur [Attiya@Wiley'04].

# Design elements of distributed storage systems

| Design Elements | Description | Effects |
|---|---|---|
| **Topology** | How are the replicas organized? | Define the system's basic complexity, availability, and efficiency. |
| **Concurrency Control** | How do the replicas interact? | Define the system's scalability and performance. |
| **Ordering Policy** | How do the replicas order operations? | Define the system's consistency level. |

**Table 5:** Design elements of distributed storage systems.

| **Topology** | | | | **Concurrency Control** | | **Ordering Policy** | |
|---|---|---|---|---|---|---|---|
| Primary-backup | Multi-master | Peer-to-peer | Chain [8] | Pessimistic | Optimistic | Ordering establishment | Conflict resolution |
| | | | | | | | |

**Table 6:** Design options for each design element.

[8] Go to Backup Slides for Chain Replication

# Consistency models and design elements

| Consistency models | Design elements | | |
|---|---|---|---|
| | **Topology** | **Concurrency control** | **Ordering policy** |
| Weak consistency models | | | |
| Strong consistency models | | | |
| Hybrid consistency models [9] | | | |

**Table 7:** Explore design elements from the perspective of weak, strong, and hybrid consistency models in distributed storage systems.

**In the following, one typical storage system per consistency.**

[9] Go to Backup Slides for Hybrid Consistency Models

# Consistency Models in DSM: Theory and Practice

# Weak consistency models

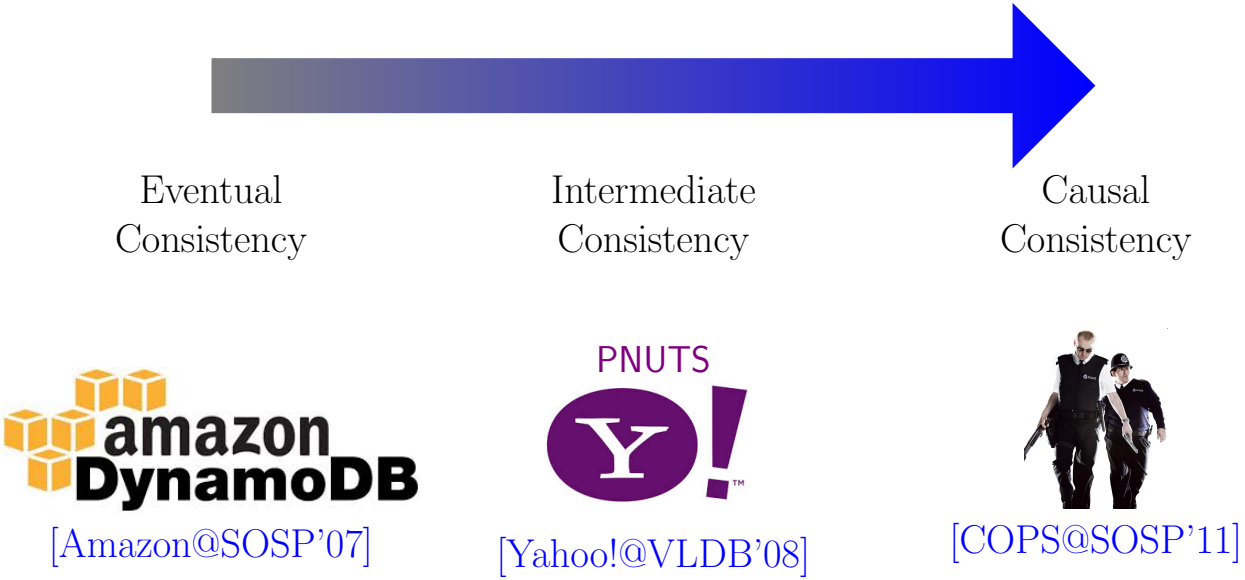



**Figure 21:** A spectrum of weak consistency models: from eventual consistency to causal consistency.

# Eventual consistency

**Definition (Eventual consistency [Terry@SOSP'95])**

Eventual consistency means that all replicas will *eventually* reach a consistent state *if* `writes` stop arriving.

Go to Backup Slides for Formal Definition of Eventual Consistency

# Case study: Dynamo at Amazon

## Rationale behind Dynamo (AP + L > C) [Amazon@SOSP'07]:

1. availability ("always-on" experience) > consistency
   - ▷ "always writable" in Shopping Cart Service
2. built for latency sensitive applications

| | Topology | | | | Concurrency Control | | Ordering Policy | |
|---|---|---|---|---|---|---|---|---|
| | Primary-backup | Multi-master | P2P | Chain | Pessimism | Optimism | Ordering establishment | Conflict resolution |
| **Dynamo** [10] | | | ✔ | | ✔ [11] | ✔ | NO specific at `writes` | Version clock upon `reads` |

**Table 8:** Design elements of Dynamo at Amazon.

[10] Go to Backup Slides for Other Case Studies for Eventual Consistency

[11] Recommended in [Amazon@SOSP'07]

# Variants of eventual consistency

| Consistency | Description |
|---|---|
| Read-your-writes | `read` operations reflect previous `writes` |
| Monotonic `reads` | successive `reads` reflect a non-decreasing set of `writes` |
| Writes-follow-reads | `writes` are propagated after `reads` on which they depend |
| Monotonic writes | `writes` are serialized by the same process |
| Session | the combination of the above |
| Consistent prefix | reader observes an ordered sequence of `writes` |
| Per-record timeline consistency | all `writes` to the same record applied in the same order |

**Table 9:** Variants of eventual consistency [Vogels@CACM'09], [Terry@CACM'13].

# Case study: PNUTS at Yahoo!

## Rationale behind PNUTS (AP + L > C) [Yahoo!@VLDB'08]:

- availability dominates
  - "If we cannot serve ads, Yahoo! does not get paid"
- strong consistency is expensive and unnecessary
- eventual consistency is too weak for Yahoo!'s applications

| | Topology | | | | Concurrency Control | | Ordering Policy | |
|---|---|---|---|---|---|---|---|---|
| | Primary-backup | Multi-master | P2P | Chain | Pessimism | Optimism | Ordering establishment | Conflict resolution |
| **PNUTS** [12] | ✔ | | | | ✔ | | Serialized versions | Not necessarily in normal cases |

**Table 10:** Design elements of PNUTS at Yahoo!.

[12] Go to Backup Slides for Case Studies for Variants of Eventual Consistency

# Consistency Models in DSM: Theory and Practice

# Strong consistency models

**Definition (Strong consistency)**

In a strong consistency condition, all processes should agree on the same view of the order in which operations occur [Attiya@Wiley'04].

**Two typical examples:**

- atomicity (external consistency)
- sequential consistency

# Case study: Megastore at Google

## Google's productions:

Chubby [Google@OSDI'06] ⇒ Bigtable [Google@OSDI'06] ⇒ **Megastore** [Google@CIDR'11]⇒ Spanner [Google@OSDI'12]

## Rationale behind Megastore:

Programmers want strong consistency.

| | Topology | | | | Concurrency Control | | Ordering Policy | |
|---|---|---|---|---|---|---|---|---|
| | Primary-backup | Multi-master | P2P | Chain | Pessimism | Optimism | Ordering establishment | Conflict resolution |
| **Megastore** [13] | | ✔ | | | ✔ | | Multi-Paxos | Not necessarily |

**Table 11:** Design elements of Megastore at Google.

[13] Go to Backup Slides for Case Study of Megastore

# Consistency Models in DSM: Theory and Practice

# Consistency Models in DSM: Theory and Practice

# Gap between theory and practice: overview

1. Gap between theory and practice
2. Gap in terms of SLA
3. Bridging gap in an SLA-guided way

# Gap between theory and practice

| In practice | Example | Theory says, |
|---|---|---|
| Claims without proofs | "We provide atomicity to users." | Verify it! |
| Trade-offs without quantitation | "If users can tolerate the inconsistencies, ..." | Quantify it! |
| Experiments without analysis | "In most cases, it makes progress well." | Formalize it! |
| Descriptions without definitions | "We care about availability, scalability, convergence" | Define it! |

**Table 12:** "Criticize" practice from the point of view of theory.

# Gap between theory and practice

| In theory | Example | Practice says, |
|---|---|---|
| Specification: safety first | "Safety " | Not in reality. |
| Lower bounds: on assumptions | "Wait-free is independent." | |
| Maintenance: on worst cases | "each `read` completes in two rounds." | |

**Table 13:** "Criticize" theory from the point of view of practice.

# Gap between theory and practice

**Question: Why do theory and practice deviate?**

**An Answer: They differ in values [Fischer@DC'03], [Zhou@SIGACT News'09].**


| Theory | Practice |
|---|---|
| abstraction | details |
| elegance | ad-hoc designs |
| for understanding | **SLA** |

**Table 14:** Theory and practice differ in values.

# Gap in terms of SLA

**Definition (Service Level Agreement (SLA) [Amazon@SOSP'07])**

SLA is a negotiated contract where a client and a service agree on several system-related characteristics.

**SLA is the specification/QOS of storage systems.**

**Theory fails in practice:**

- SLA prevails in industry
- SLA in practice $\gg$ specification in theory

# Service Level Agreement (SLA)

**An example of SLA [Amazon@SOSP'07]:**

**Client:** peak load of 500 requests per second

**Service:** responds within 300ms for 99.9% of its requests

**Another example of SLA [Terry@SOSP'13]:**

*I'd ideally like to be able to*

1. *read my own writes, but*
2. *I'll accept any consistency as long as data is returned in under 300 ms*

# SLA in practice versus Specification in theory

## SLA in practice vs. Specification in theory

1. More new ones, **e.g.,**
   - availability
   - scalability
2. Finer-grained, **e.g.,**
   - quantification of consistency
   - most cases vs. rare cases (99.9%)
   - latency in "real-time" (*ms*)

# Consistency Models in DSM: Theory and Practice

# Bridging Gap in an SLA-guided Way

**Bridging Gap in an SLA-guided Way:**

1. verification of consistency models
2. quantification of consistency models
3. other possible directions

# Verifying consistency models

**Definition (Verifying a consistency model)**

- $\mathcal{I}$nstance:
  - a *read/write* execution $e$
  - a consistency model $\mathcal{C}$
- $\mathcal{P}$roblem:
  - Does the execution $e$ satisfy the consistency model $\mathcal{C}$?

**We overview the algorithms and complexity.**

$n \triangleq$ **# of operations in** $e$.

# Verifying consistency models: VSC and VL

**VSC: Verifying Sequential Consistency; VL: Verifying Linearizability**

| Variants | VSC results | VL results |
|---|---|---|
| general problem | NP-complete | NP-complete |
| 2 operations per process | NP-complete | w.l.o.g. [14] |
| 2 variables | NP-complete | w.l.o.g. |
| 3 processes | NP-complete | $O(n \log n)$ |
| read-mapping | NP-complete | $O(n \log n)$ |
| write-order | NP-complete | $O(n \log n)$ |
| `read&write` only | NP-complete | NP-complete |
| conflict-order | $O(n \log n)$ | $O(n \log n)$ |

**Table 15:** A summary of results on verifying sequential consistency (**VSC**) and linearizability (**VL**) [Gibbons@SIAM J. Comput.'97].

[14] The complexity is not affected by the given restriction.

# Verifying consistency models: VMC

## VMC: Verifying Memory Coherence (a.k.a Cache Consistency)

| Variants | Read/Write | Read-Modify-Write |
|---|---|---|
| 1 Operation/Process | $O(n \lg n)$ | $O(n^2)$ |
| 2 Operations/Process | ? | NP-complete |
| 3+ Operations/Process | NP-complete | NP-complete |
| Constant $k$ Processes | $O(n^k)$ | $O(n^k)$ |
| 1 Write/Value (*read-mapping*) | $O(n)$ | $O(n \lg n)$ |
| 2 Writes/Value | NP-complete | ? |
| 3+ Writes/Value | NP-complete | NP-complete |
| Write-order Given | $O(n^2)$ | $O(n)$ |

**Table 16:** A summary of complexity results for verifying memory coherence (**VMC**) [Cantin@SPAA'03], [Cantin@TPDS'05].

# Verifying consistency models

**Verifying Safety, Regularity, and Atomicity:**

1. Variant: write unique value
2. Online: detect a consistency violation as soon as one happens

| | **Safety** | **Regularity** | **Atomicity** | **Sequential** |
|---|---|---|---|---|
| **Offline** [Anderson@HotDep'10] | $O(n^2)$ | $O(n^2)$ | $O(n^3)$ | *not studied* |
| **Online** [15] [Golab@PODC'11] | $O(n)$ | $O(n)$ | $O(n \log n)$ | Poly$(n)$ |

$O(n \log n)$ for 2-atomicity verification [Golab@ICDCS'13].

[15] With some other assumptions.

# Verifying consistency models: VPC

## VPC: Verifying PRAM Consistency

| | *(S)ingle variable* | *(M)ultiple variables* |
|---|---|---|
| *write (D)uplicate values* | VPC-SD<br>(NPC) [$*$] | VPC-MD<br>(NPC) [$*$] |
| *write (U)nique value* | VPC-SU<br>(P) [Golab@PODC'11] | VPC-MU<br>(P) [$*$] |

**Table 17:** A summary of complexity results for VPC problem ([$*$] : new results).

# Quantifying consistency models

**Definition (Quantifying a consistency model)**

- $\mathcal{I}$nstance:
  - a *read/write* execution $e$
  - a consistency model $\mathcal{C}$
- $\mathcal{P}$roblem:
  - How much does the execution $e$ (not) satisfy the consistency model $\mathcal{C}$? (**Beyond 0/1.**)

# Quantifying consistency models

## Quantifying consistency models from four perspectives

**[Yu@TOCS'02]:**

1. data versions
2. randomness
3. timeliness Go to Backup Slides for Timeliness
4. numerical values Go to Backup Slides for Numerical Values

# Quantifying consistency models: data versions

**Specification:**

definitions of $k$-safe, $k$-regular, and $k$-atomic [Aiyer@DISC'05], [Taubenfeld@ICDCN'13], [Jagadeesan@ICALP'14]

- ▷ `reads` are allowed to return stale values

**Maintenance:**

- ▷ $k$-quorum systems [Aiyer@DISC'05]

# Quantifying consistency models: data versions

## Computability power and limitations [Taubenfeld@ICDCN'13]

**Theorem**

*$K$-safe registers ($k \geq 1$) is computationally equivalent to 1-atomic registers.*

**Proof.**

By simulations. □

**Theorem**

*SWMR $k$-safe registers can solve mutual exclusion (and $l$-exclusion).*

# Quantifying consistency models: randomness

**Definition ((Single-writer) random register [Lee@DC'05])**

1. Random register: eventually each write stops being read from
2. $P$-random register
3. Monotone random register

**Maintenance:**

Using probabilistic quorum systems [Malkhi@PODC'97].

# Quantifying consistency models: randomness

**Question: What is the computability power of random registers [Lee@DC'05]?**

- ▷ definition of multi-writer random registers
- ▷ simulations
- ▷ power (and limitations) of solving other problems

**One possible future work.**

# Quantifying consistency models: randomness

**Probabilistically Bounded Staleness (PBS) [Bailis@VLDB'12] is a way to quantify eventual consistency:**

1. How eventual?
2. How consistent?

# Quantifying consistency models: randomness

**Almost strong consistency:**

- deterministically **bounded staleness**

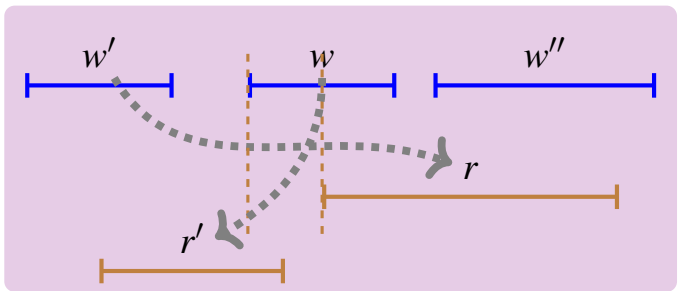



**Figure 22:** The case for `read` operation `r` to get *2-stale* value.

- **probabilistically** quantifying the occurrence of "reading stale values"

$$\mathbb{P}(\texttt{2-atomicity}) = \mathbb{P}(\texttt{CP} \cap \texttt{ONI}) = \mathbb{P}(\texttt{CP}) \cdot \mathbb{P}(\texttt{ONI} \mid \texttt{CP}).$$

# Formal study of weak consistency models

## Understanding eventual consistency

1. Principles of Eventual Consistency [Burckhardt@NOW Publishers'14]
2. Eventually Linearizable Shared Objects [Serafini@PODC'10]

## Data types for eventual consistency

1. Replicated Data Types: Specification, Verification, Optimality [Burckhardt@POPL'14]
2. Conflict-free Replicated Data Types [Shapiro@SSS'11]
3. Semantics of Eventually Consistent Replicated Sets [Bieniusa@DISC(BA)'12]

# Availability, Latency, and Progress

## Availability

1. Minimal Replication Cost for Availability [Yu@PODC'02]
2. Availability in Globally Distributed Storage Systems [Google@OSDI'10]

## Latency

1. Consistency or Latency? A Quantitative Analysis of Replication Systems Based on Replicated State Machines [Wang@DSN'13]
2. Refined Quorum Systems [Guerraoui@PODC'07]

## Progress

1. Obstruction-Free Algorithms Can Be Practically Wait-free [Fich@DISC'05]
2. Are Lock-free Concurrent Algorithms Practically Wait-free? [Alistarh@STOC'14]

# Consistency Models in DSM: Theory and Practice

# Conclusion

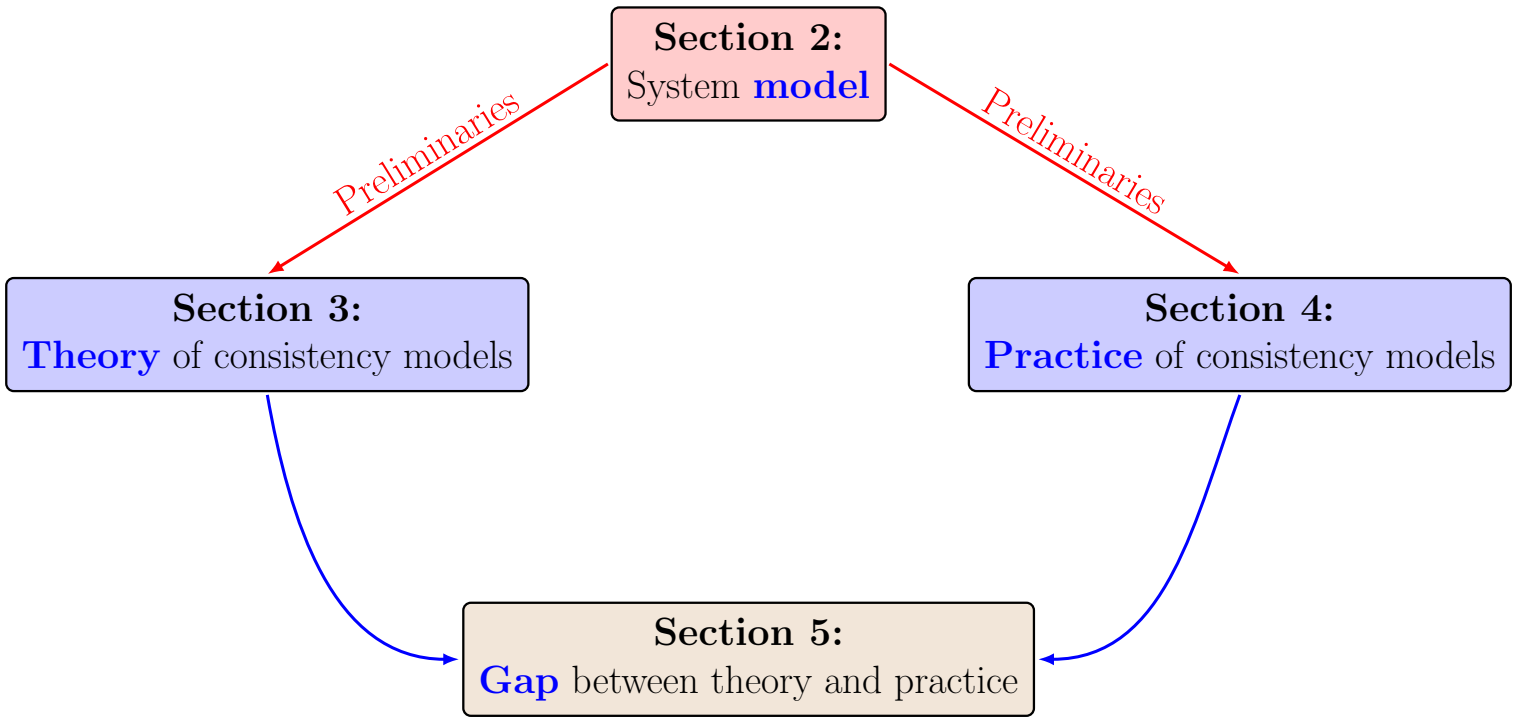



**Figure 23:** Contents of the survey on consistency models in dsm.


THANK YOU

# Consistency Models in DSM: Theory and Practice

# Atomicity as an example



Each operation appears to take effect instantaneously at some moment between its `invocation` and `response`.

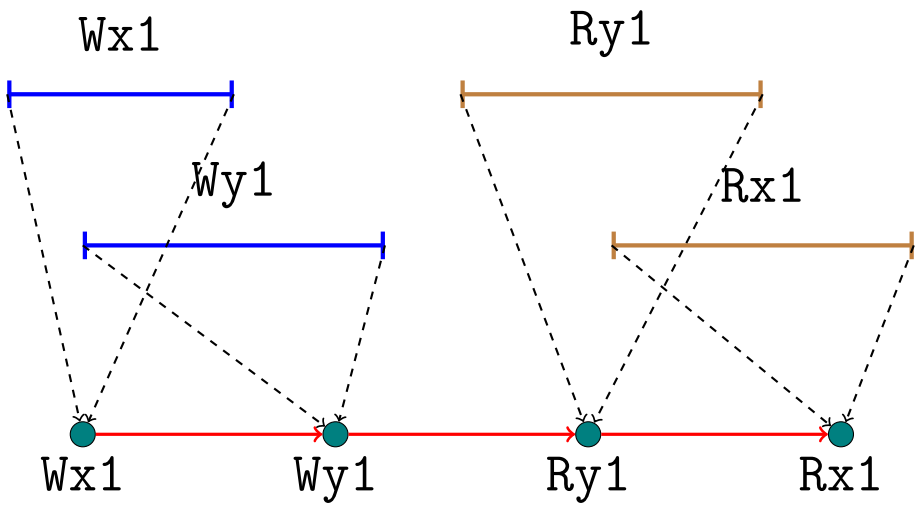



**Figure 24:** An atomic execution.

**Atomicity $\equiv$ `read` semantics + real-time order**

# Sequential consistency as an example



Each operation appears to take effect in program order.

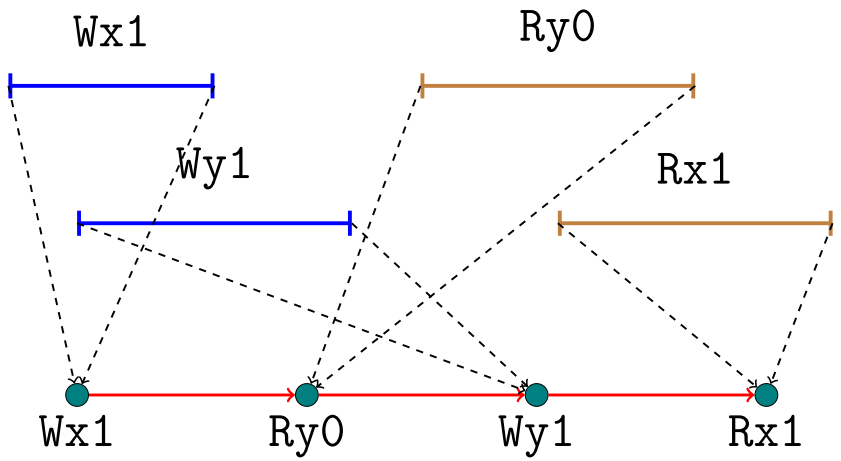



**Figure 25:** A sequential (but not atomic) execution.

**Sequential consistency $\equiv$ `read` semantics + program order**

# Maintaining atomic registers

Back to Maintenance

1. The ABD algorithm for single-writer atomic registers

   [ABD@JACM'95], [Attiya@EATCS'10]

   **Timestamps:** *local counter* for the single writer

   **Quorum:** `read-write` overlapping $\Rightarrow$ "old-new inversions"

   - *reader* writes back + $\forall_{R_1,R_2 \in \mathcal{R}} : R_1 \cap R_2 \neq \emptyset$

2. Algorithm for multi-writer atomic registers [Lynch@FTCS'97]

   **Timestamps:** *(local counter, pid)* for all writers

   **Quorum:** `write-write` overlapping $\Rightarrow$ (QUES: what?)

   - *writer* reads before writing + $\forall_{R \in \mathcal{R}, W \in \mathcal{W}} : R \cap W \neq \emptyset$

# Maintaining safe registers

1. Algorithm for single-writer safe registers [Lamport@DC'86]
2. Algorithm for multi-writer safe registers [Not yet well-defined]

# Maintaining regular registers

1. Algorithm for single-writer regular registers [Lamport@DC'86]
2. Algorithm for multi-writer regular registers [Shao@SIAM J. Comput.'11]

# Maintaining sequential consistency

(QUES: Quorum???)

Maintaining sequential consistency [Attiya@TOCS'94], [Fekete@JACM'98]

**Timestamps:** totally ordered broadcast

**Quorum:**

- local `read`: $|R \in \mathcal{R}| = 1, |W \in \mathcal{W}| = n$
- local `write`: $|R \in \mathcal{R}| = n, |W \in \mathcal{W}| = 1$

# Maintaining causal consistency

Maintaining causal consistency [Ahamad@DC'95]

**Timestamps:** vector clock based on the "happened-before" relation [Lamport@CACM'78]

**Quorum:**

- local `read`: $|R \in \mathcal{R}| = 1, |W \in \mathcal{W}| = n$

# Maintaining PRAM consistency

Maintaining PRAM consistency [Lipton@TR'88]

# Let's move to the impossibility results

Go to Impossibility Results

# Impossibility results for atomic registers

**Theorem ([Attiya?@Wiley'04])**

*It is impossible to simulate single-writer multi-reader, atomic register in message passing if more than half replicas can crash ($f > \frac{n}{2}$).*

**Proof.**

By "partition-merger". □

# Impossibility results for atomic registers

**Theorem ([Attiya@TOCS'94])**

*It is impossible to have local read or local write algorithms for atomicity as long as there is any uncertainty in the message delay.*

**Proof.**

□

# Impossibility results for atomic registers

**Theorem ([Dutta@PODC'04])**

*It is impossible to obtain a fast* [16] *simulation of single-writer multi-reader, atomic register tolerating* $f < \frac{n}{2}$ *replica crashes.*

**Proof.**

By "block execution" argument. □

**Theorem ([Dutta@PODC'04])**

*If any number of readers and writers may fail, no fast simulation of multi-writer multi-reader, atomic registers can tolerate the failure of even one replica.*

[16] Both *reads* and *writes* complete in a single round-trip.

# Impossibility results for sequential consistency

**Theorem**

# More impossibility results

paper “Possibility and Impossibility Results in a Shared Memory Environment”

# Let's move to the lower bounds

Go to Lower Bounds

# Adding time to the system model

**To obtain lower bounds on the time to perform `reads/writes`:**

1. local clocks run at the same rate; not initially synchronized
2. message delay $\in [d - u, d]$
   - $d$: upper bound
   - $u$: uncertainty

# Lower bounds

**Theorem**

*pram*

# Lower bounds

**Theorem ([Attiya@TOCS'94])**

*For any sequentially consistent memory consistency system that provides two read/write objects, $t_{read} + t_{write} \geq d$.*

**Theorem ([Attiya@TOCS'94])**

*In any linearizable implementation of an object, the worst-case time is*

- $t_{write} \geq \frac{u}{2}$
- $t_{read} \geq \frac{u}{4}$

**Proof.**

By "shifting". □

# Consistency Models in DSM: Theory and Practice

# Eventual consistency

**Definition (Formal definition of eventual consistency [Serafini@PODC'10])**

**Nontriviality:**

**Set stability:**

**Prefix consistency:**

# Case study for eventual consistency

Dynamo, Porcupine, Bayou, Project Voldemort

Go to Case Study of Dynamo at Amazon

# Case study: Dynamo at Amazon

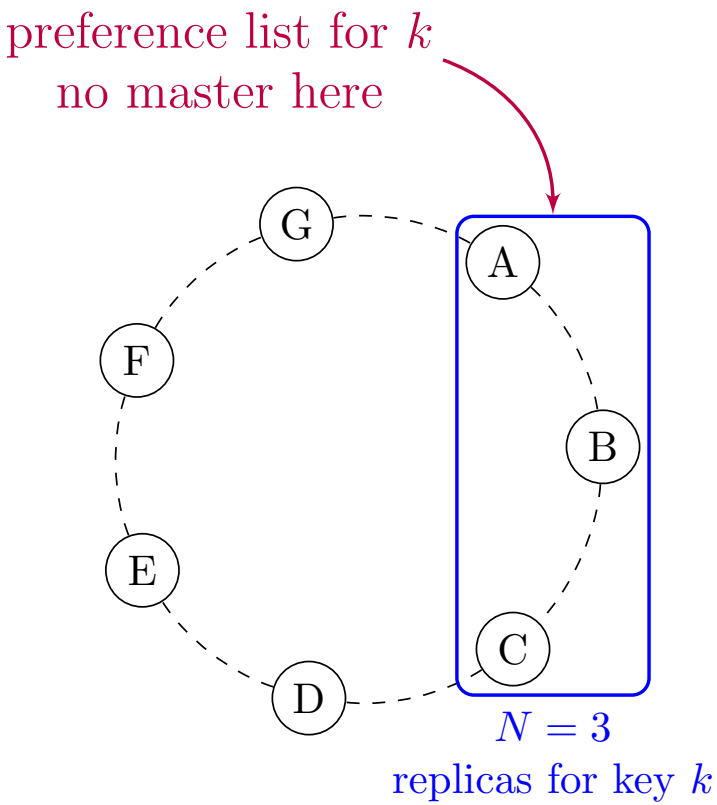
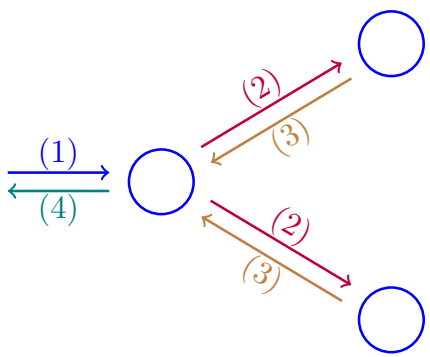



**Figure 26:** Topology: replicas of keys in Dynamo ring without master.

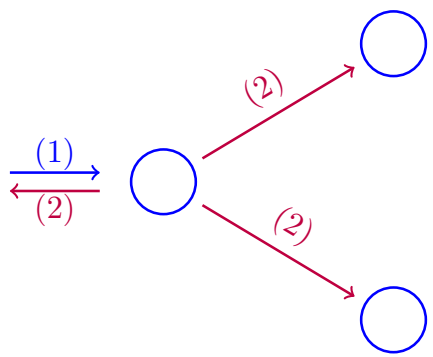
# Case study: Dynamo at Amazon

## Configurable "sloppy quorum":

- pessimistic: $(N, R, W) = (3, 2, 2)$ in [Amazon@SOSP'07]
- optimistic: $R = 1$, $W = 1$




(a) Pessimistic concurrency control for $(R, W, N) = (3, 2, 2)$.




(b) Optimistic concurrency control for $R = 1$ or $W = 1$.

**Figure 27:** Concurrency control: configurable sloppy quorum.

# Case study: Dynamo at Amazon

## Data versioning for conflict resolution:

- ▷ no specific operation orders to keep upon `writes`
- ▷ syntactic and semantic resolution (upon a `read`)

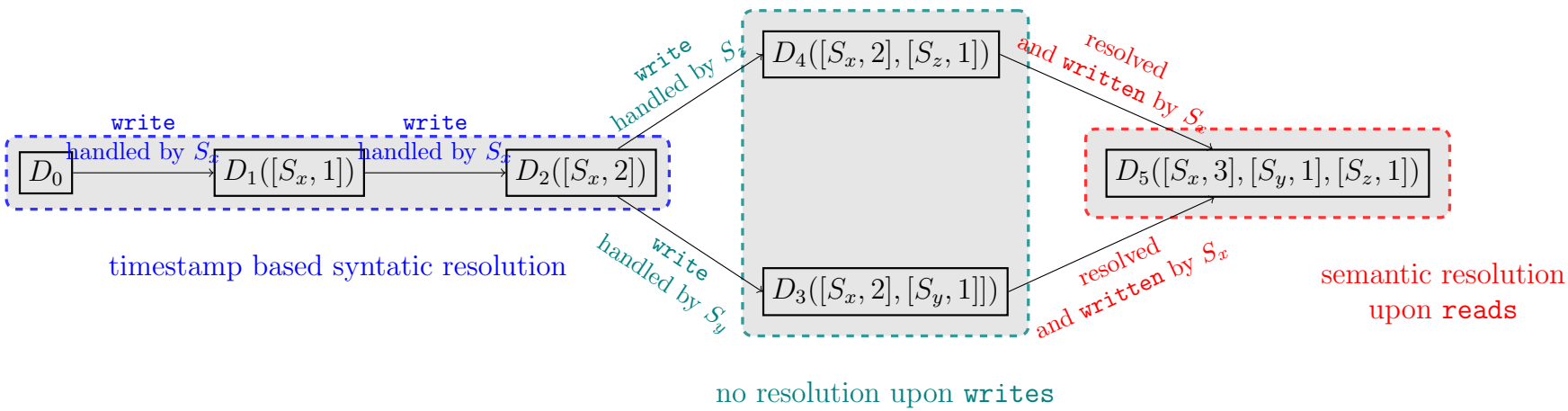



**Figure 28:** Syntatic and semantic conflict resolution using vector clock *(node, counter)*.

# Case study: PNUTS at Yahoo!

record-level timeline consistency enforced through record mastership [PNUTS@Internet Computing'12]

Go to Case Study of PNUTS at Yahoo!

# Case study of variants of eventual consistency

Tao at Facebook: "read-after-write": It is optimized heavily for reads, and explicitly favors efficiency and availability over consistency.

# Case study: COPS at Princeton

Go to Case Study of COPS at Princeton

# Case study: Megastore at Google

Go to Case Study of Megastore at Google

# Case study: Spanner at Google

Scalable consistency in scatter

Scalable, distributed data structures for internet service construction: linearizable key-value store tolerant of churn.

# Case study: Windows Azure at Microsoft

# Case study: ZooKeeper at Apache

# Case study: chain replication

[Renesse@OSDI'04] Go to Design Elements

# Hybrid consistency models

Go to Consistency Models and Design Elements

eventually serializable; eventually linearizable

# Consistency Models in DSM: Theory and Practice

# Verifying consistency models

**In the context of shared memory multiprocessor:**

| Consistency model | Complexity | Ref. |
|---|---|---|

**Table 18:** An *incomplete* list of results on verifying consistency models in the context of shared memory multiprocessor.

# Quantifying consistency models: timeliness

[Rojas@PODC'99] Go to Quantifying Consistency Models

# Quantifying consistency models: timeliness

# Quantifying consistency models: timeliness

# Quantifying consistency models: numerical values

Go to Quantifying Consistency Models