

# 复制数据类型理论研究

## (青年学者学术沙龙)

魏恒峰

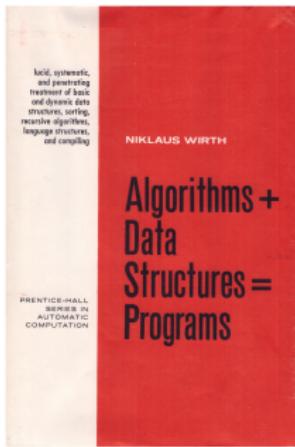
南京大学软件所

2018 年 12 月 21 日



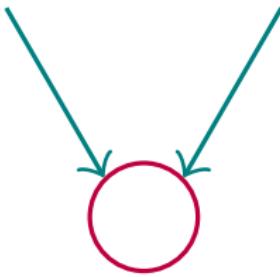
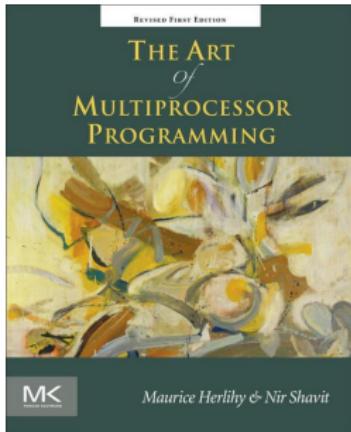
# Abstract Data Types [Liskov and Zilles, 1974]

## (单线程; 顺序语义)



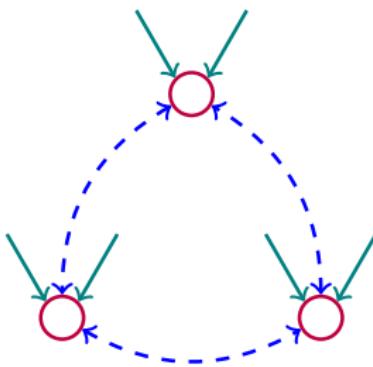
# Concurrent Data Types [Herlihy and Wing, 1990]

(多线程; 并发语义)



PL (Programming Language)

## Replicated Data Types [Shapiro et al., 2011a] [Burckhardt et al., 2014] (多副本; 复制语义)

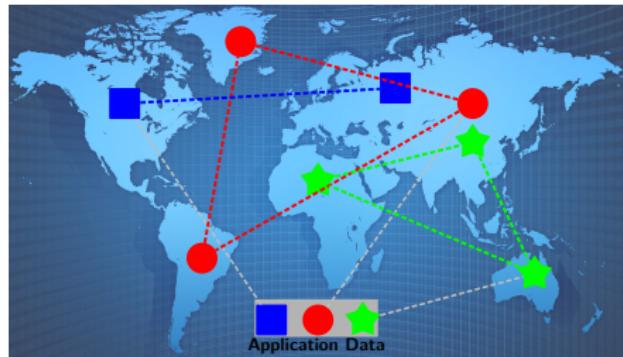


DC (Distributed Computing)

## 新平台：大规模分布式系统



低延迟    高可用性 (4 个 9)    高容错性    高可扩展性



## 分布数据 (distributed data):

1. 分区 (partition): 水平扩展
2. 副本 (replication) : 就近访问, 容灾备份

## 复制数据类型 [Shapiro et al., 2011a]

- ▶ Read/Write Register
- ▶ Counter
- ▶ Set
- ▶ List
- ▶ HashMap
- ▶ Disjoint Set
- ▶ Graph
- ▶ ...

What's  
new?

新问题, 新挑战

## Replicated Data Types: Specification, Verification, Optimality

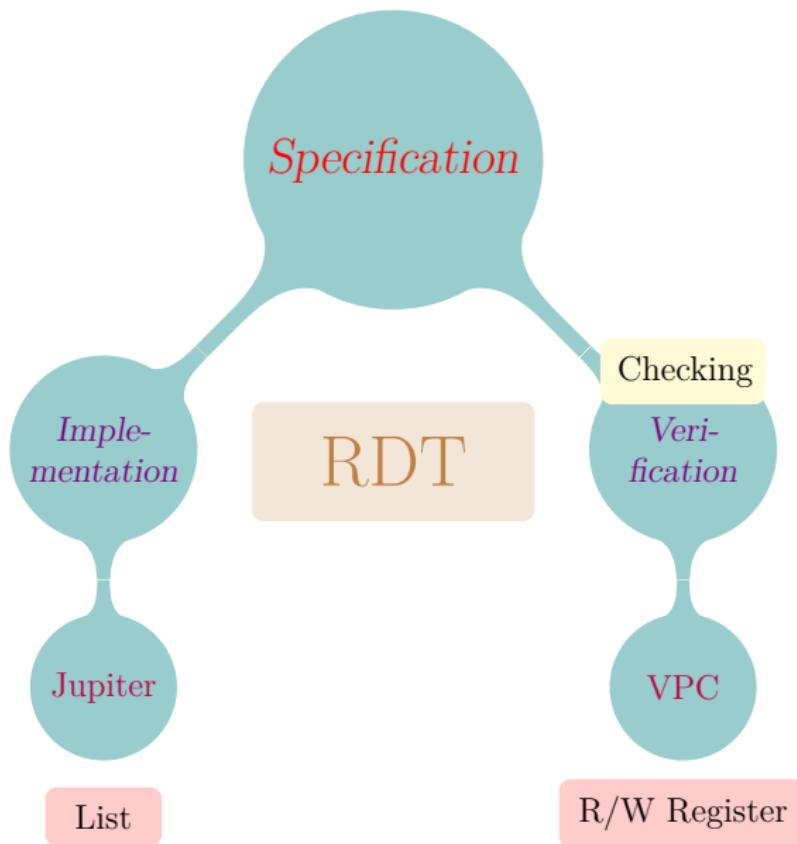
Sebastian Burckhardt

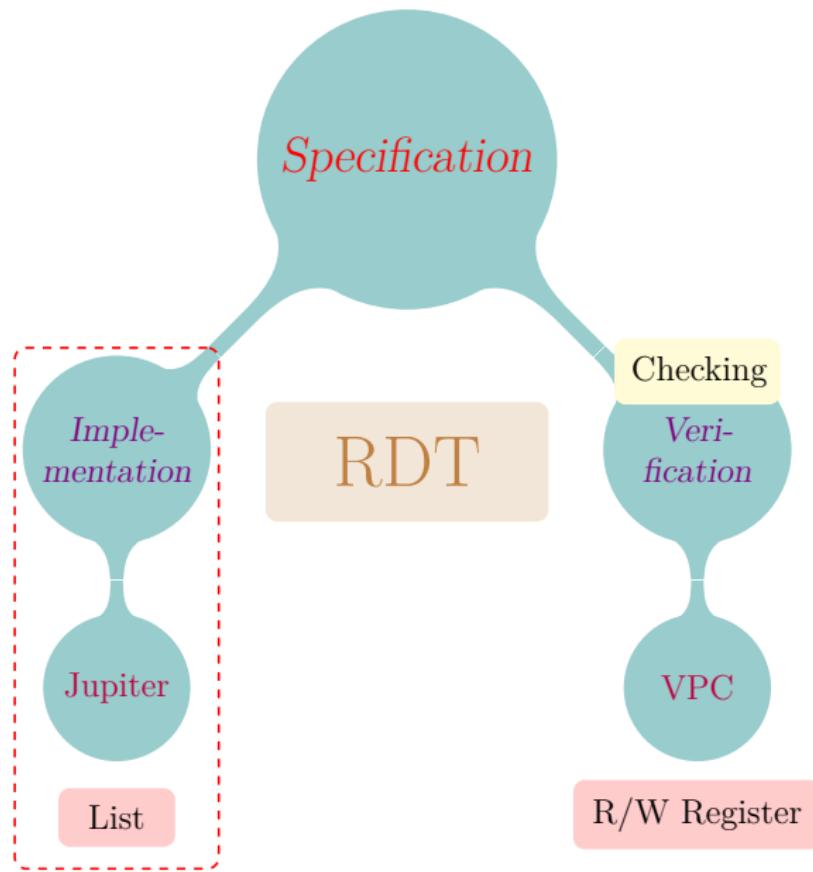
Alexey Gotsman

Hongseok Yang

Marek Zawirski

[Burckhardt et al., 2014]





## 复制列表数据类型<sup>1</sup>

实现复制列表的 Jupiter 协议 [Nichols et al., 1995]<sup>a</sup> 满足 weak list specification [Attiya et al., 2016]<sup>b</sup>.

---

<sup>a</sup>David A. Nichols et al. (1995). “High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System”. In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST ’95. ACM, pp. 111–120.

<sup>b</sup>Hagit Attiya et al. (2016). “Specification and complexity of collaborative text editing”. In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC ’16. ACM, pp. 259–268.

---

<sup>1</sup>藏在脚注里的猜想@PODC’2016 [Attiya et al., 2016]



## 提纲

1. 为什么要研究复制列表数据类型?
2. 什么是 weak list specification  $\mathcal{A}_{\text{weak}}$ ?
3. Jupiter 协议是如何工作的?
4. 如何证明 Jupiter 满足  $\mathcal{A}_{\text{weak}}$ ?

# 基于副本的协同文本编辑系统



(a) Google Docs



(b) Apache Wave



(c) Wikipedia



(d) LATEX Editor

- ▶ 副本节点要**立即**响应本地用户操作
- ▶ 更新操作**异步**传播到其它副本节点

# 复制列表对象：建模编辑系统的核心功能

$\text{INS}(a, p)$ ：在  $p$  位置插入元素  $a$

$\text{DEL}(p)$ ：删除  $p$  位置上的元素

$\text{READ}$ ：返回该列表

定义 (最终收敛性 (Eventual Convergence) [Ellis and Gibbs, 1989])

当系统处于“静默”(Quiescence)状态时，所有副本节点上的列表是相同的。

定义 (强最终一致性 (Strong Eventual Consistency) [Shapiro et al., 2011b])

如果两个副本节点处理了同一组更新操作，则它们的列表是相同的。

对系统的中间状态缺少足够的约束

## Specification and Complexity of Collaborative Text Editing

Hagit Attiya  
Technion

Sebastian Burckhardt  
Microsoft Research

Alexey Gotsman  
IMDEA Software Institute

Adam Morrison  
Technion

Hongseok Yang  
University of Oxford

Marek Zawirski<sup>\*</sup>  
Inria & Sorbonne Universités,  
UPMC Univ Paris 06, LIP6

定义 (Weak List Specification  $\mathcal{A}_{\text{weak}}$  [Attiya et al., 2016])

*Informally,  $\mathcal{A}_{\text{weak}}$  requires the ordering between elements that are not deleted to be consistent across the system.*

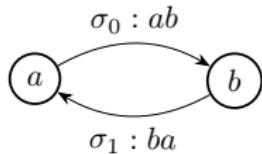
定义在系统所有列表状态上的全局性质

$\mathcal{A}_{\text{weak}}$  等价于:

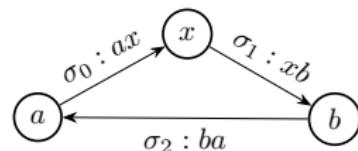
定义 (状态对兼容性) (Pairwise State Compatibility Property)

任给两个列表状态  $\sigma_0, \sigma_1$ , 若它们含有两个共同元素  $a, b$ ,  
则  $a, b$  在  $\sigma_0$  与  $\sigma_1$  中的相对顺序保持一致。

$\sigma_0 : ab$        $\sigma_1 : ba$



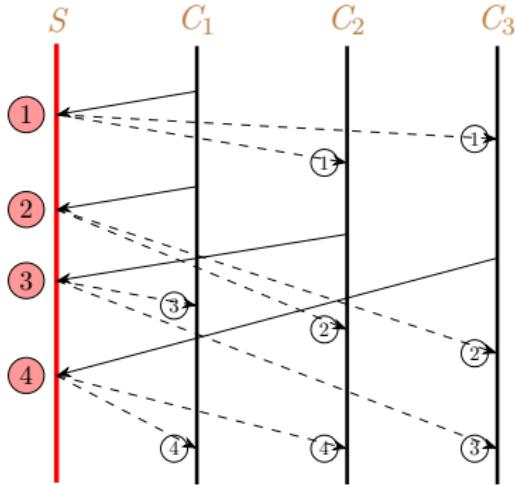
$\sigma_0 : ax$        $\sigma_1 : xb$        $\sigma_2 : ba$



“Strong List Specification” 不允许

# Jupiter

$(n+1)$  replicas  $\triangleq$   $(n)$  Client +  $(1)$  Server [Nichols et al., 1995]

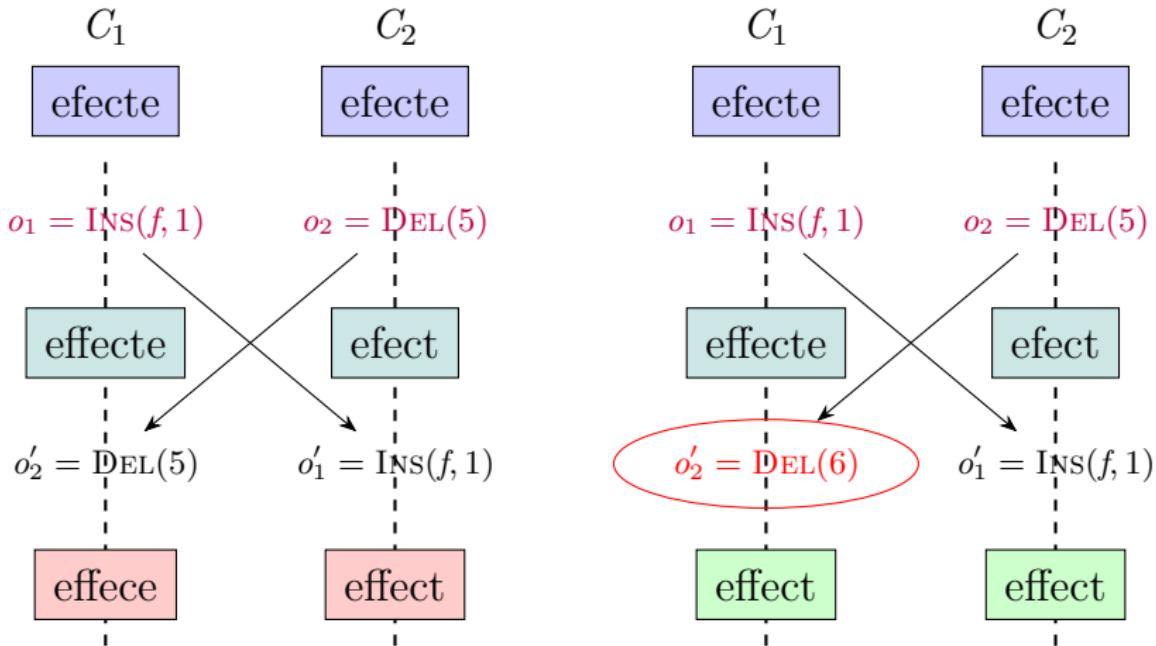


Server 负责将所有操作序列化



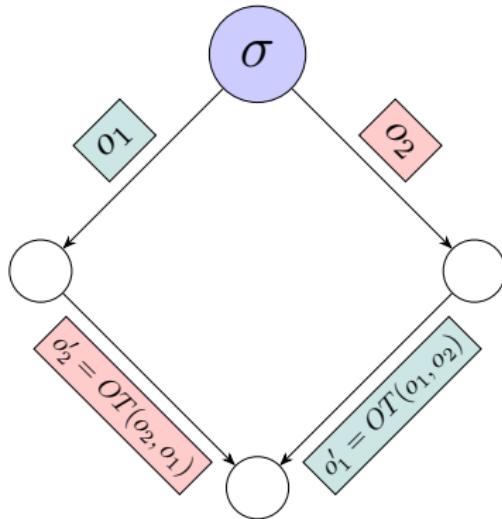
# 挑战: 如何处理并发操作带来的冲突?

(The server is not drawn.)



解决方案: 操作转换 (Operational Transformation; OT)

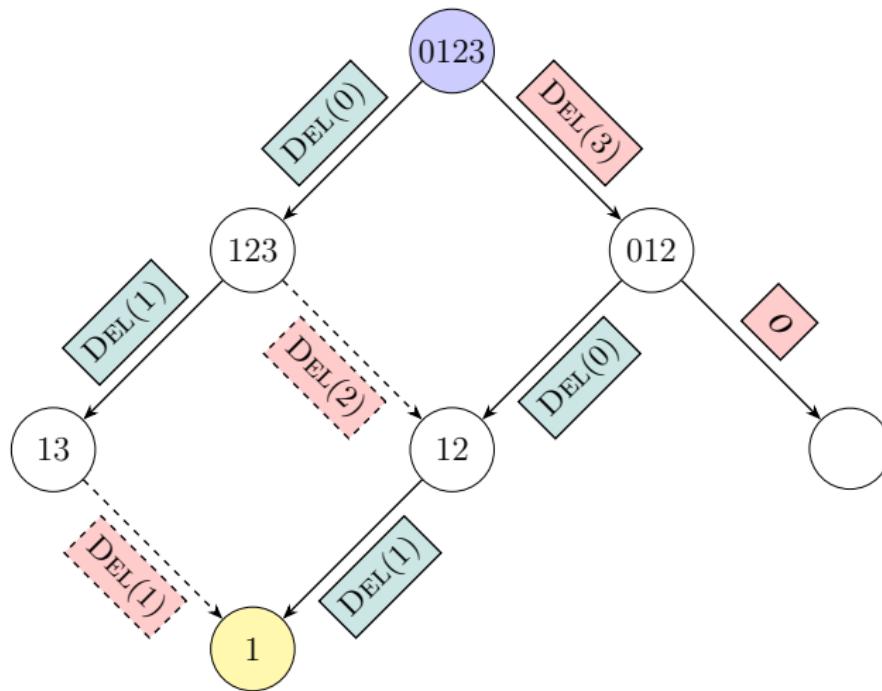
[Ellis and Gibbs, 1989]



交换律:  $\sigma; o_1; o'_2 \equiv \sigma; o_2; o'_1$

[Ellis and Gibbs, 1989]

问题: 如果副本节点之间偏离 (diverge) 多个 ( $\geq 2$ ) 操作, 怎么办?



### 核心问题:

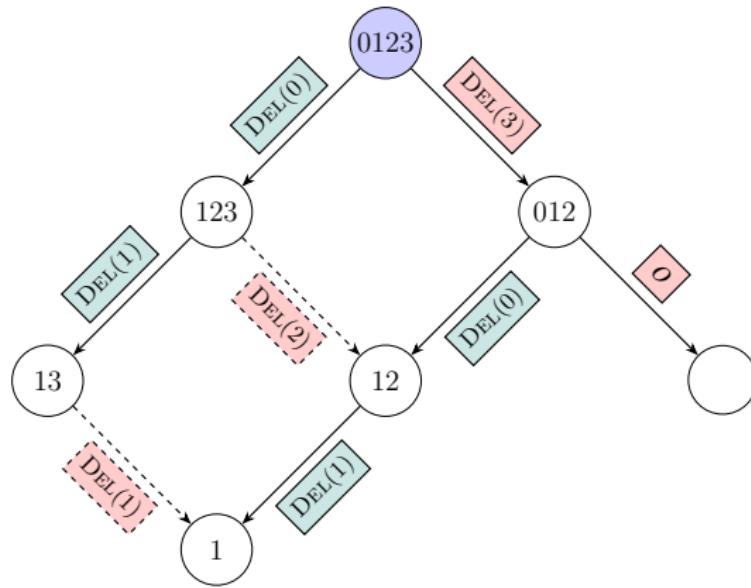
当副本节点  $r$  接收到由 Server 转发的操作  $o$  时，  
如何对  $o$  执行操作转换？

与哪些操作进行转换？以什么顺序进行转换？

### 核心思想:

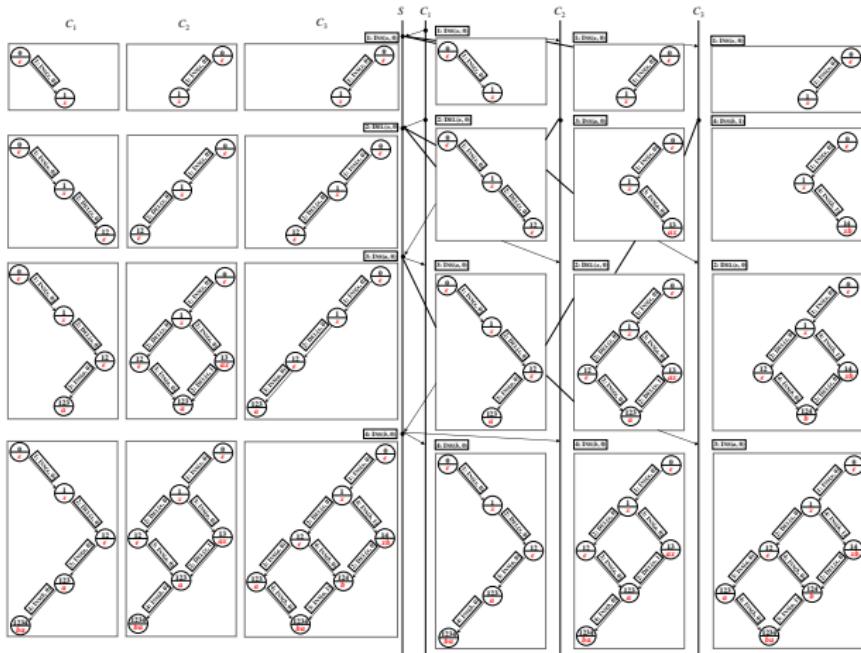
1. 与 “ $r$  上已执行且与  $o$  并发的操作” 进行转换
2. 以 “由 Server 确定的序列化顺序” 进行转换

利用数据结构 **2D 状态空间** [Xu, Sun, and Li, 2014]  
控制何时以及如何执行“操作转换”



2D: LOCAL vs. GLOBAL

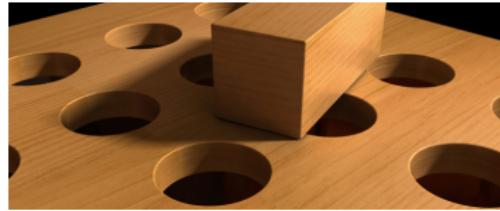
## 每个 Client 维护一个 2D 状态空间



Server 维护  $n$  个 2D 状态空间, 与  $n$  个 Clients 对应

# Mismatch!

$\mathcal{A}_{\text{weak}}$  所规定的全局性质



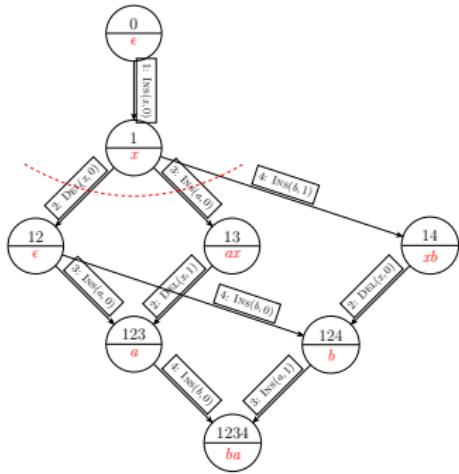
Jupiter 协议中, 每个副本节点所维护的局部视图

# CJupiter (Compact Jupiter)

## 定理 (等价性)

在相同的操作调度 (schedule) 下, CJupiter 与 Jupiter 中的对应副本节点的行为 (behavior; 状态序列) 是相同的。

CJupiter 为每个副本节点维护一个 ***n*-ary 有序状态空间**

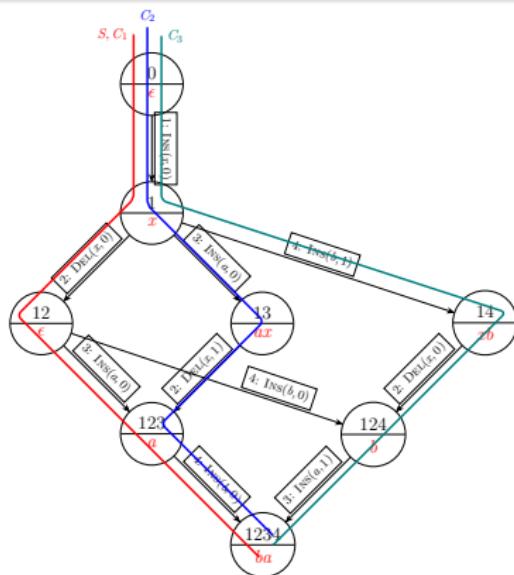


每个 node 可以有  $> 2$  条出边

每个 node 的所有出边按照其上标记操作的“序列化”顺序排成全序

## 命题 (Compactness of CJupiter)

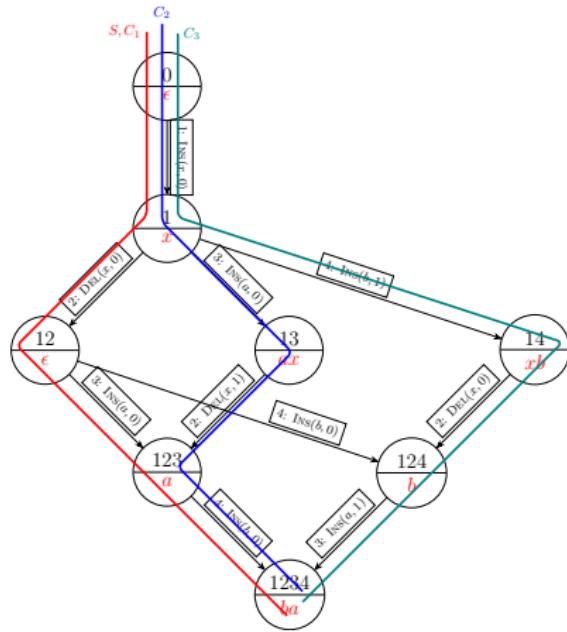
CJupiter 所维护的  $(n + 1)$  个  $n$ -ary 有序状态空间是相同的。



每个副本节点的行为对应于该状态空间中的一条 **路径**  
 单个  $n$ -ary 有序状态空间包含了系统所有可能的状态

# CJupiter 满足 Weak List Specification

关注某个  $n$ -ary 有序状态空间, 三步骤 证明“状态对兼容性”



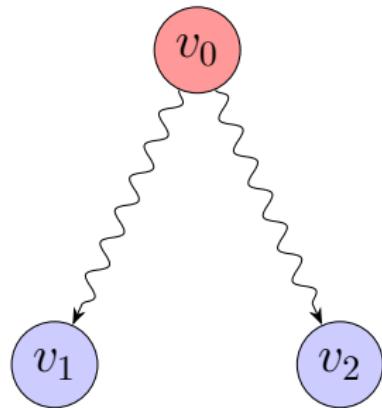
反证法、数学归纳法、分情形分析法

1

任取两个状态节点  $v_1$  和  $v_2$

引理 (LCA (Lowest Common Ancestor))

$n$ -ary 有序状态空间中的任意一对状态节点都有唯一的最近公共祖先。

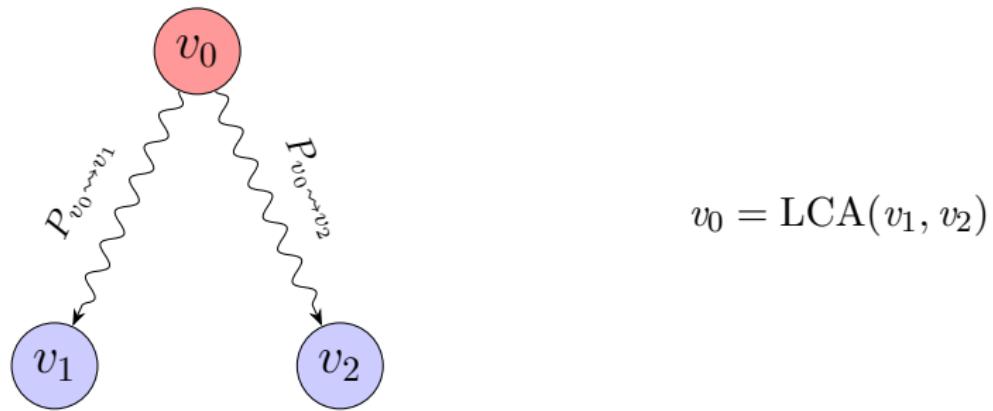


$$v_0 = \text{LCA}(v_1, v_2)$$

2 考虑从  $v_0 = \text{LCA}(v_1, v_2)$  到  $v_1$  和  $v_2$  的两条路径

### 引理 (Disjoint Paths)

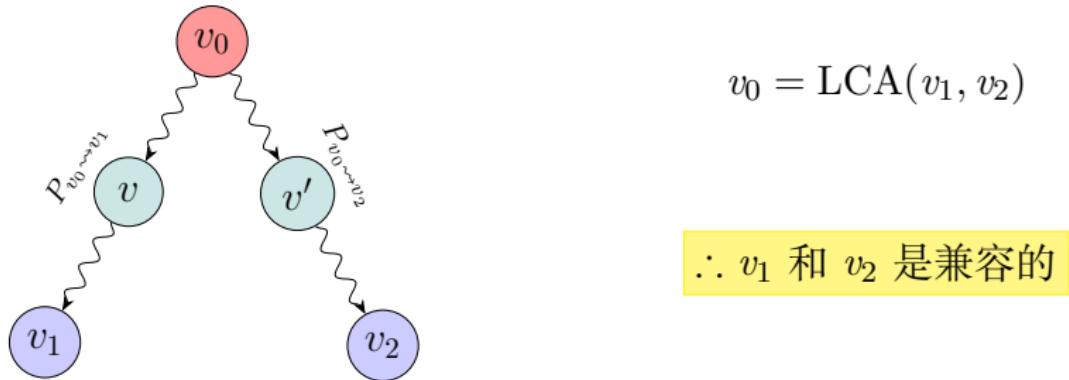
路径  $P_{v_0 \rightsquigarrow v_1}$  上包含的操作集  $O_{v_0 \rightsquigarrow v_1}$  与路径  $P_{v_0 \rightsquigarrow v_2}$  上包含的操作集  $O_{v_0 \rightsquigarrow v_2}$  不相交。



### 3 考虑两条路径上的状态

#### 引理 (Compatible Paths)

$P_{v_0 \rightsquigarrow v_1}$  上的任一状态  $v$  与  $P_{v_0 \rightsquigarrow v_2}$  上的任一状态  $v'$  是兼容的。



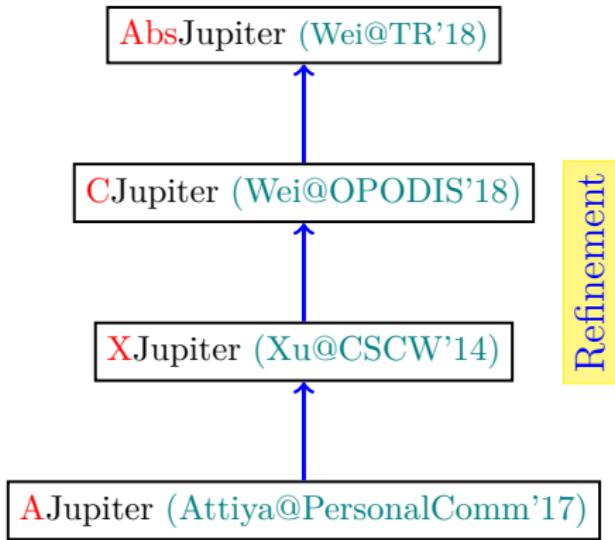
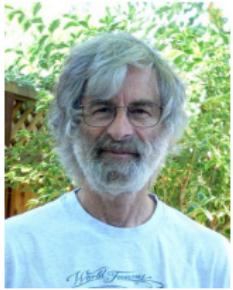
个人体会: 基于 OT 思想的协议晦涩难懂

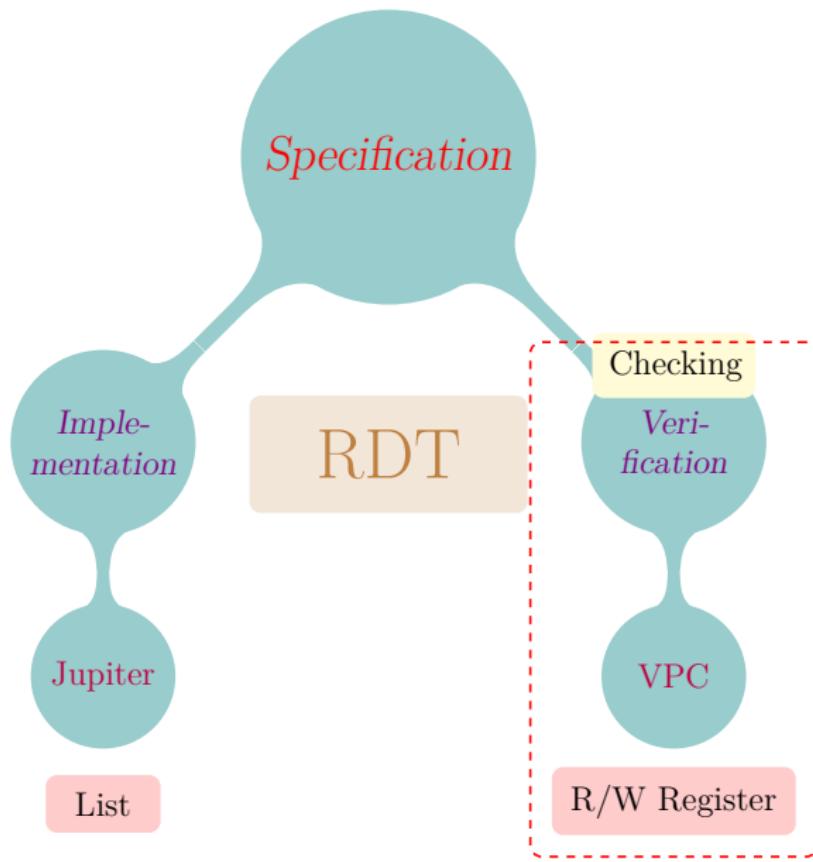


- ▶ 协议多种多样
- ▶ 经常不加证明
- ▶ 证明是错误的
- ▶ 勘误也是错的

# Model Checking/Theorem Proving: 使用 TLA+/TLAPS

jupiter-tlaplus@github





## 协议验证 (Verification of a Protocol)

[Bouajjani, Enea, and Hamza, 2014] [Bouajjani et al., 2017]

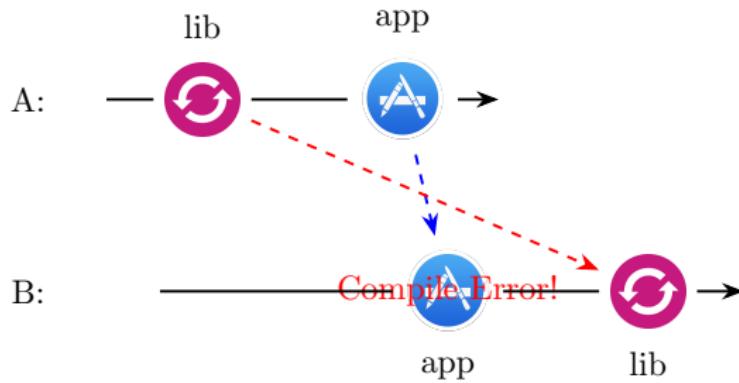
执行验证 (Verification of an Execution)



黑盒测试/确认系统是否提供了其所声称的数据一致性

[DeCandia et al., 2007] [Golab, Li, and Shah, 2011]

**PRAM:** 包含存储系统常提供的最基本的“会话”(session)一致性  
[Terry et al., 1994] [Brzezinski, Sobaniec, and Wawrzyniak, 2004]



PRAM 保证“单调写”性质

## 定义 (VPC (Verifying PRAM Consistency) 判定问题)

实例: 系统执行 (execution  $e$ )

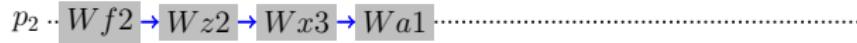
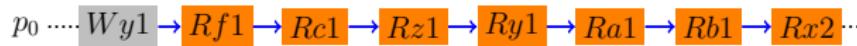
问题: 该执行  $e$  是否满足 PRAM 一致性模型 ( $\mathcal{C}$ )?

$$e \in \mathcal{C} \Rightarrow \{0, 1\}?$$

## 定义 (系统执行)

系统执行  $e \triangleq \{h_p \mid h_p : \text{进程 } p \text{ 上的读写操作序列}\}$

规模  $n$ : 系统执行中读写操作的总数

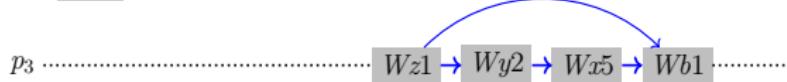
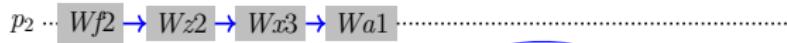
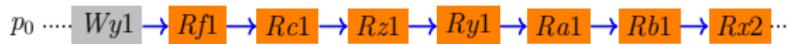


## 定义 (PRAM 一致性模型)

系统执行  $e$  满足 PRAM 一致性



$\forall p : p$  上所有操作与其它进程上所有写操作存在合法调度



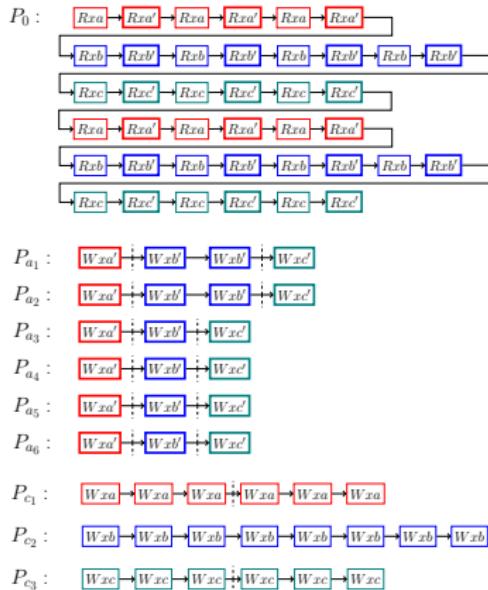
$p_0 : Wf2 \ Wf1 \ Wz2 \ \color{blue}{Wz1} \ Wy2 \ Wy1 \ \color{brown}{Rf1} \ Wx5 \ Wx3 \ \color{red}{Wx2} \ Wc1 \ \color{brown}{Rcl}$   
 $Rz1 \color{brown}{Ry1} \color{blue}{Wa1} \ Ra1 \color{blue}{Wb1} \ Rb1 \ Rx2$

VPC 问题的四种变体 (按“执行”的类型) 及复杂度  
([\*] : 本文工作)

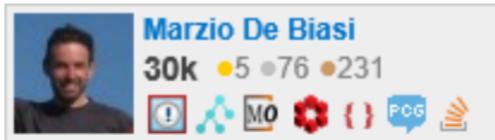
	<i>(S)ingle variable</i>	<i>(M)ultiple variables</i>
<i>write (D)uplicate values</i>	VPC-SD <b>(NP-complete) [*]</b>	VPC-MD <b>(NP-complete) [*]</b>
<i>write (U)nique value</i>	VPC-SU <b>(P)</b> [Golab, Li, and Shah, 2011]	VPC-MU <b>(P) [*]</b>

Read-mapping [Gibbons and Korach, 1997]:  $\forall r, \exists! w, f(r) = w$ .

# VPC-SD (VPC-MD) 是 NP-complete 问题



UNARY 3-PARTITION 实例  $A = \{2, 2, 1, 1, 1, 1\}$ ,  $m = 2$ ,  $B = 4$  对应的 VPC-SD 执行

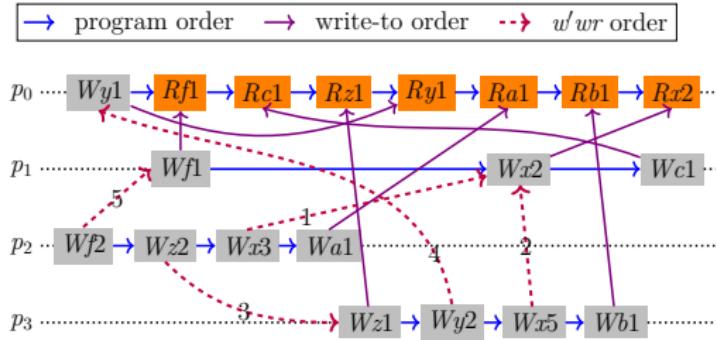


*“Basically I'm a programmer :-)*

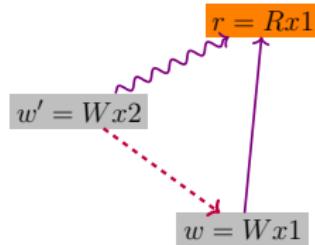
*I enjoy reading papers about the complexity of puzzle games,  
and I'm writing (amateur) proofs on the complexity of a few  
puzzle games.”*



## VPC-MU 的多项式算法 RW-CLOSURE



RW-CLOSURE 算法示例：在传递闭包之上迭代应用  $w'wr$  规则



## 定理 (RW-CLOSURE 算法正确性)

VPC-MU 实例满足 PRAM 一致性



RW-CLOSURE 算法所得图是 DAG 图

证明

“ $\implies$ ” 反证法

“ $\impliedby$ ” 对读操作作数学归纳，构造合法调度

RW-CLOSURE 算法复杂度：

$$\underbrace{O(n^2)}_{\# \text{loops}} \cdot \underbrace{O(n^3)}_{\text{transitive closure}} = O(n^5)$$

RW-CLOSURE 算法的缺点:

- ▶ 在全图上应用  $w'wr$  规则
- ▶ 应用  $w'wr$  规则无特定顺序

VPC-MU 的多项式算法 READ-CENTRIC 要点:

- ▶ 增量式调度每个读操作
- ▶ 在读操作诱导的局部子图上按逆拓扑序应用  $w'wr$  规则

## 定理 (READ-CENTRIC 算法正确性)

VPC-MU 实例满足 PRAM 一致性



READ-CENTRIC 算法所得图是 DAG 图

### 证明

$$\text{READ-CENTRIC} \iff^{\text{Reachability}} \text{RW-CLOSURE}$$

READ-CENTRIC 算法复杂度:

$$\underbrace{O(n)}_{\# \text{ reads}} \cdot \underbrace{O(n \cdot n^2)}_{\text{TOPO-SCHEDULE}} = O(n^4)$$

**VSC** (Verifying Sequential Consistency) 与 **VL** (Verifying Linearizability)  
问题的复杂度 [Gibbons and Korach, 1997]

Variants	VSC	VL
General	NP-complete	NP-complete
2 Operations/Process	NP-complete	NP-complete
2 Variables	NP-complete	NP-complete
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	$O(n \log n)$	$O(n \log n)$

**VMC** (Verifying Memory Coherence) 问题的复杂度 [Cantin, Lipasti, and Smith, 2005]

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	$O(n^2)$	$O(n)$

## Atomicity<sup>2</sup> 相关一致性模型验证问题复杂度 (假设: 不允许写重复值)

	Safety	Regularity	Atomicity	Sequential
<b>Offline</b> [Anderson et al., 2010]	$O(n^2)$	$O(n^2)$	$O(n^3)$	<i>not studied</i>
<b>Online<sup>3</sup></b> [Golab, Li, and Shah, 2011]	$O(n)$	$O(n)$	$O(n \log n)$	$\text{Poly}(n)$

---

<sup>2</sup>也称 Linearizability

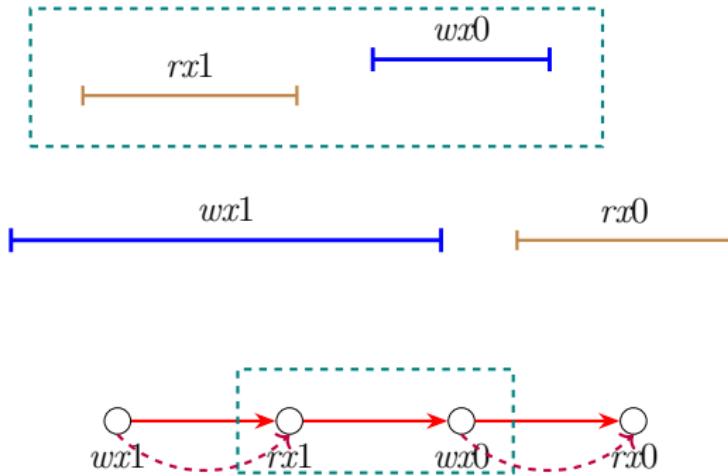
<sup>3</sup>包含其它假设

## $k$ -AV ( $k$ -Atomicity Verification) 问题复杂度

Problems	Variants	Results	Work
1-AV	General	NP-complete	[Gibbons and Korach, 1997]
1-AV	Write unique value	$O(n \log n)$	[Gibbons and Korach, 1997]
2-AV	Write unique value	$O(n \log n)$	[Golab, Hurwitz, and Li, 2013]
$k$ -AV	Write unique value	$O(n^2)$	[Golab et al., 2015]
	Bounded concurrency		[Golab et al., 2018]
$k$ -AV	Write unique value		

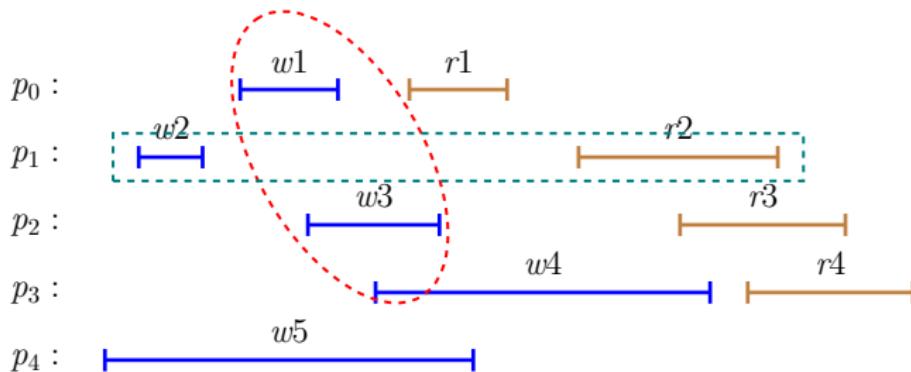
Atomicity = 实时序 + 读写语义

[Lamport, 1986]



$k$ -Atomicity = 实时序 +  $k$ -读写语义

[Aiyer, Alvisi, and Bazzi, 2005] [Taubenfeld, 2013]

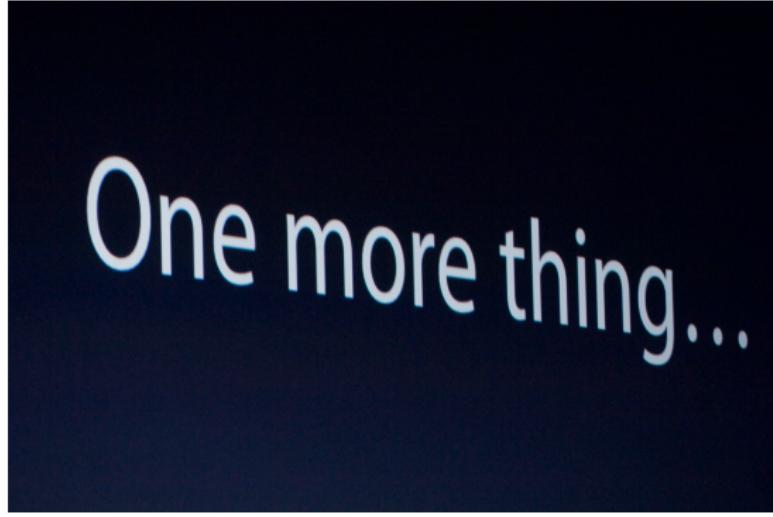


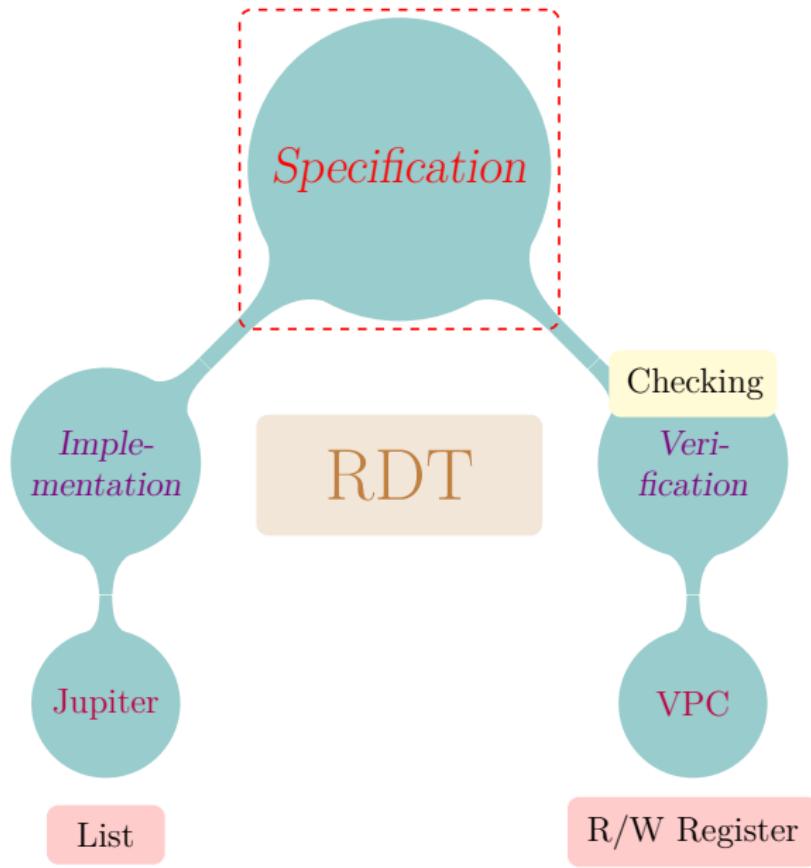
3-Atomicity :  $w5 \quad w2 \quad w1 \quad r1 \quad w3 \quad w4 \quad r2 \quad r3 \quad r4$

定义 ( $k$ -AV ( $k$ -Atomicity Verification) 判定问题)

实例：系统执行  $e$  (不允许写重复值)、参数  $k$

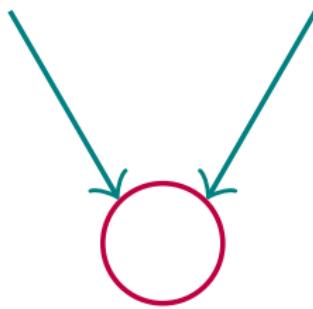
问题：该执行  $e$  是否满足  $k$ -Atomicity?



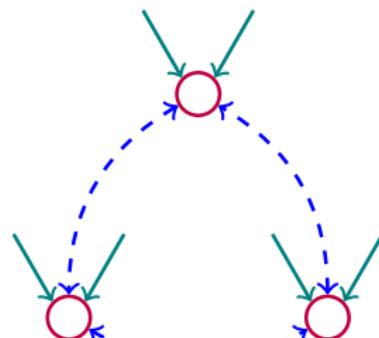


## 规约: 数据一致性模型 (Consistency Model)

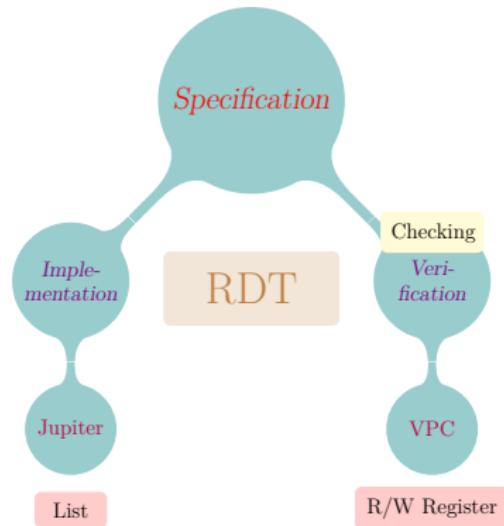
多处理器系统中的并发数据类型



分布式系统中的复制数据类型



PL + DC + FM [Burckhardt et al., 2014]



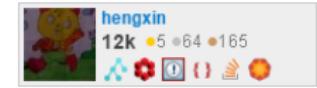
# Thank You!

hfwei@nju.edu.cn



[hengxin@homepage](mailto:hengxin@homepage)

[hengxin@github](https://github.com/hengxin)



[hengxin@stackexchange](https://stackexchange.com/users/121451/hengxin)

## (I) DC + PL + FM:

- ▶ 有哪些问题可以共享?
- ▶ 有哪些理论、技术可以相互借鉴?
- ▶ 分布式系统中的编程模型?
- ▶ 并发数据类型的验证技术应用于复制数据类型?
- ▶ 分布式系统中的数据一致性模型 *vs.* 多处理器系统中的内存模型?
- ▶ ...

## (II) 研究经验分享:

- ▶ 如何选择研究问题?
- ▶ 给青年学者的建议
- ▶ ...

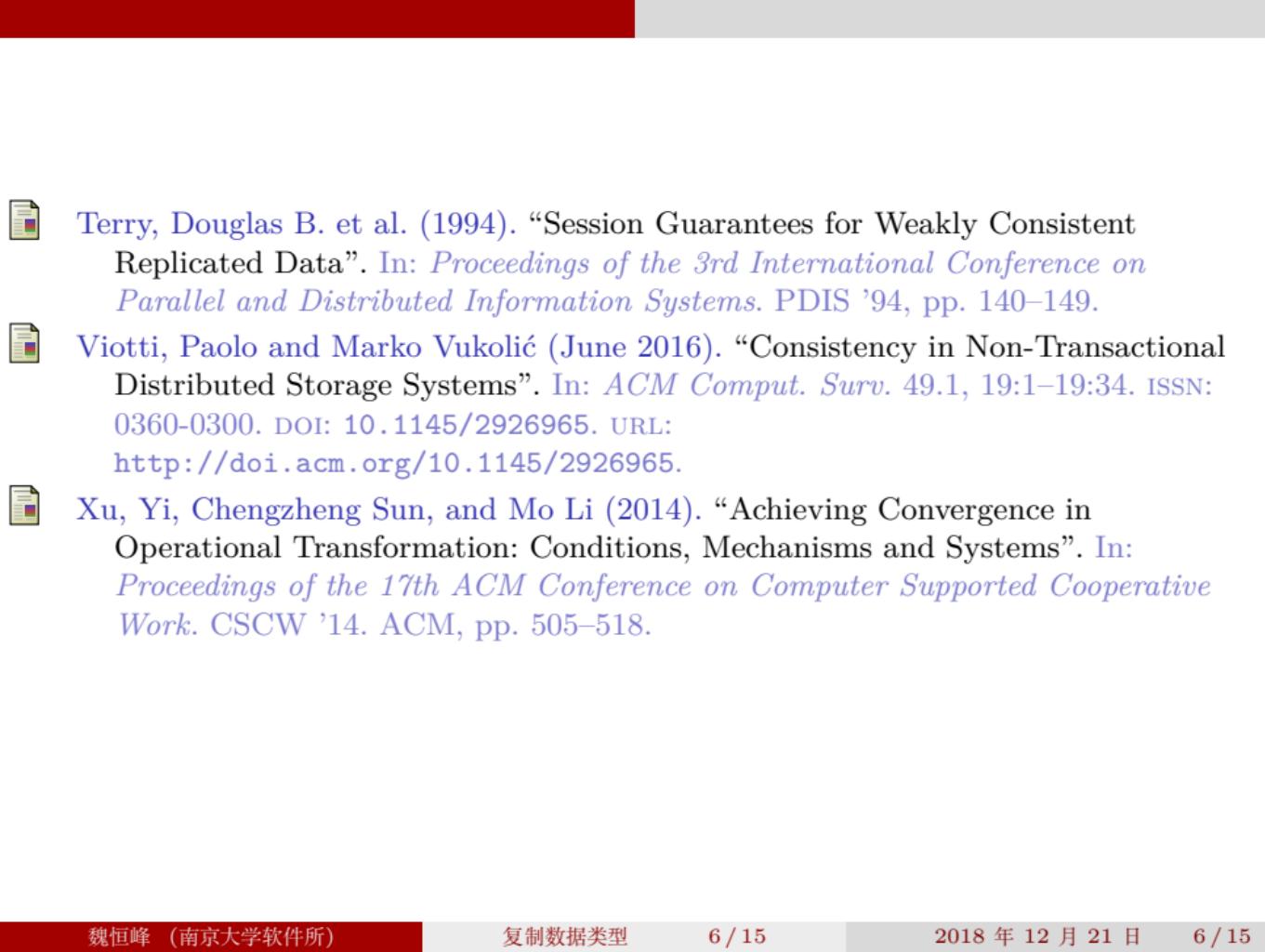
-  Aiyer, Amitanand, Lorenzo Alvisi, and Rida A. Bazzi (2005). “On the Availability of Non-strict Quorum Systems”. In: *Proceedings of the 19th International Conference on Distributed Computing*. DISC '05. Springer-Verlag, pp. 48–62.
-  Anderson, Eric et al. (2010). “What Consistency Does Your Key-value Store Actually Provide?” In: *Proceedings of the Sixth International Conference on Hot Topics in System Dependability*. HotDep'10. Vancouver, BC, Canada: USENIX Association, pp. 1–16. URL:  
<http://dl.acm.org/citation.cfm?id=1924908.1924919>.
-  Attiya, Hagit et al. (2016). “Specification and complexity of collaborative text editing”. In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. ACM, pp. 259–268.
-  Bailis, Peter et al. (2012). “Probabilistically Bounded Staleness for Practical Partial Quorums”. In: *Proc. VLDB Endow.* 5.8, pp. 776–787.
-  Bouajjani, Ahmed, Constantin Enea, and Jad Hamza (2014). “Verifying Eventual Consistency of Optimistic Replication Systems”. In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. San Diego, California, USA: ACM, pp. 285–296. ISBN: 978-1-4503-2544-8. DOI: 10.1145/2535838.2535877. URL:  
<http://doi.acm.org/10.1145/2535838.2535877>.

-  Bouajjani, Ahmed et al. (2017). "On Verifying Causal Consistency". In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. Paris, France: ACM, pp. 626–638. ISBN: 978-1-4503-4660-3. DOI: 10.1145/3009837.3009888. URL: <http://doi.acm.org/10.1145/3009837.3009888>.
-  Brzezinski, J, C Sobaniec, and D Wawrzyniak (2004). "From session causality to causal consistency". In: *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 152–158.
-  Burckhardt, Sebastian (Oct. 2014). "Principles of Eventual Consistency". In: *Found. Trends Program. Lang.* 1.1-2, pp. 1–150. ISSN: 2325-1107. DOI: 10.1561/2500000011. URL: <http://dx.doi.org/10.1561/2500000011>.
-  Burckhardt, Sebastian et al. (2014). "Replicated Data Types: Specification, Verification, Optimality". In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. San Diego, California, USA: ACM, pp. 271–284. ISBN: 978-1-4503-2544-8. DOI: 10.1145/2535838.2535848. URL: <http://doi.acm.org/10.1145/2535838.2535848>.
-  Cantin, Jason F., Mikko H. Lipasti, and James E. Smith (July 2005). "The complexity of verifying memory coherence and consistency". In: *IEEE Transactions on Parallel and Distributed Systems* 16.7, pp. 663–671.

-  Chatterjee, Shankha and Wojciech Golab (2017). “Brief Announcement: A Probabilistic Performance Model and Tuning Framework for Eventually Consistent Distributed Storage Systems”. In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC ’17. Washington, DC, USA: ACM, pp. 259–261. ISBN: 978-1-4503-4992-5. doi: 10.1145/3087801.3087850. URL: <http://doi.acm.org/10.1145/3087801.3087850>.
-  DeCandia, Giuseppe et al. (2007). “Dynamo: Amazon’s Highly Available Key-value Store”. In: *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*. SOSP ’07. ACM, pp. 205–220.
-  Ellis, C. A. and S. J. Gibbs (1989). “Concurrency Control in Groupware Systems”. In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’89. ACM, pp. 399–407.
-  Gibbons, Phillip B. and Ephraim Korach (Aug. 1997). “Testing shared memories”. In: *SIAM J. Comput.* 26.4, pp. 1208–1244.
-  Golab, Wojciech, Jeremy Hurwitz, and Xiaozhou (Steve) Li (2013). “On the k-Atomicity-Verification Problem”. In: *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems*. ICDCS ’13. IEEE Computer Society, pp. 591–600.

-  Golab, Wojciech, Xiaozhou Li, and Mehul A. Shah (2011). “Analyzing Consistency Properties for Fun and Profit”. In: *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. PODC ’11. ACM, pp. 197–206.
-  Golab, Wojciech et al. (2015). “Computing Weak Consistency in Polynomial Time: [Extended Abstract]”. In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. PODC ’15. ACM, pp. 395–404.
-  Golab, Wojciech et al. (Jan. 2018). “Computing  $k$ -Atomicity in Polynomial Time”. In: 47, pp. 420–455.
-  Herlihy, Maurice P. and Jeannette M. Wing (July 1990). “Linearizability: A Correctness Condition for Concurrent Objects”. In: *ACM Trans. Program. Lang. Syst.* 12.3, pp. 463–492. ISSN: 0164-0925. DOI: 10.1145/78969.78972. URL: <http://doi.acm.org/10.1145/78969.78972>.
-  Lamport, Leslie (1986). “On interprocess communication (Part II: algorithms)”. In: *Distrib. Comput.* 1.2, pp. 86–101.
-  Lee, Hyunyoung and Jennifer L. Welch (2005). “Randomized Registers and Iterative Algorithms”. In: *Distrib. Comput.* 17.3, pp. 209–221.

-  Liskov, Barbara and Stephen Zilles (1974). “Programming with Abstract Data Types”. In: *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages*. Santa Monica, California, USA: ACM, pp. 50–59. DOI: 10.1145/800233.807045. URL: <http://doi.acm.org/10.1145/800233.807045>.
-  Nichols, David A. et al. (1995). “High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System”. In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. ACM, pp. 111–120.
-  Shapiro, Marc et al. (Jan. 2011a). *A comprehensive study of Convergent and Commutative Replicated Data Types*. Research Report RR-7506. Inria – Centre Paris-Rocquencourt ; INRIA, p. 50. URL: <https://hal.inria.fr/inria-00555588>.
-  Shapiro, Marc et al. (2011b). “Conflict-free Replicated Data Types”. In: *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*. SSS'11. Springer-Verlag, pp. 386–400.
-  Taubenfeld, Gadi (2013). “Weak Read/Write Registers”. In: *Distributed Computing and Networking*. Ed. by Davide Frey et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 423–427. ISBN: 978-3-642-35668-1.



## 针对列表的操作转换函数 [Ellis and Gibbs, 1989]

$$OT(INS(a_1, p_1, pr_1), INS(a_2, p_2, pr_2)) = \begin{cases} INS(a_1, p_1, pr_1) & p_1 < p_2 \\ INS(a_1, p_1 + 1, pr_1) & p_1 > p_2 \\ NOP & p_1 = p_2 \wedge a_1 = a_2 \\ INS(a_1, p_1 + 1, pr_1) & p_1 = p_2 \wedge a_1 \neq a_2 \wedge pr_1 > pr_2 \\ INS(a_1, p_1, pr_1) & p_1 = p_2 \wedge a_1 \neq a_2 \wedge pr_1 \leq pr_2 \end{cases}$$
$$OT(INS(a_1, p_1, pr_1), DEL(\_, p_2, pr_2)) = \begin{cases} INS(a_1, p_1, pr_1) & p_1 \leq p_2 \\ INS(a_1, p_1 - 1, pr_1) & p_1 > p_2 \end{cases}$$
$$OT(DEL(\_, p_1, pr_1), INS(a_2, p_2, pr_2)) = \begin{cases} DEL(\_, p_1, pr_1) & p_1 < p_2 \\ DEL(\_, p_1 + 1, pr_1) & p_1 \geq p_2 \end{cases}$$
$$OT(DEL(\_, p_1, pr_1), DEL(\_, p_2, pr_2)) = \begin{cases} DEL(\_, p_1, pr_1) & p_1 < p_2 \\ DEL(\_, p_1 - 1, pr_1) & p_1 > p_2 \\ NOP & p_1 = p_2 \end{cases}$$

## 定理 (RW-CLOSURE 算法正确性)

VPC-MU 实例满足 PRAM 一致性



RW-CLOSURE 算法所得图是 DAG 图

### 证明

“ $\implies$ ” 反证法

“ $\impliedby$ ” 难点: DAG 图蕴含着多个全序

技巧: 对读操作作数学归纳, 构造合法调度

### RW-CLOSURE 算法复杂度:

$$\underbrace{O(n^2)}_{\# \text{loops}} \cdot \underbrace{O(n^3)}_{\text{transitive closure}} = O(n^5)$$

READ-CENTRIC 算法复杂度:

$$\underbrace{O(n)}_{\text{iterations}} \cdot \underbrace{O(n \cdot n^2)}_{\text{TOPO-SCHEDULE}} = O(n^4)$$

引理 (TOPO-SCHEDULE 的非迭代性)

设 TOPO-SCHEDULE 正在处理读操作  $r$ ,  
则局部子图中的每个写操作最多只有一次机会  
在满足规则  $w'wr$  的三元组中扮演 “ $w'$  角色”。

# 实验评估

实验目的<sup>1</sup>：

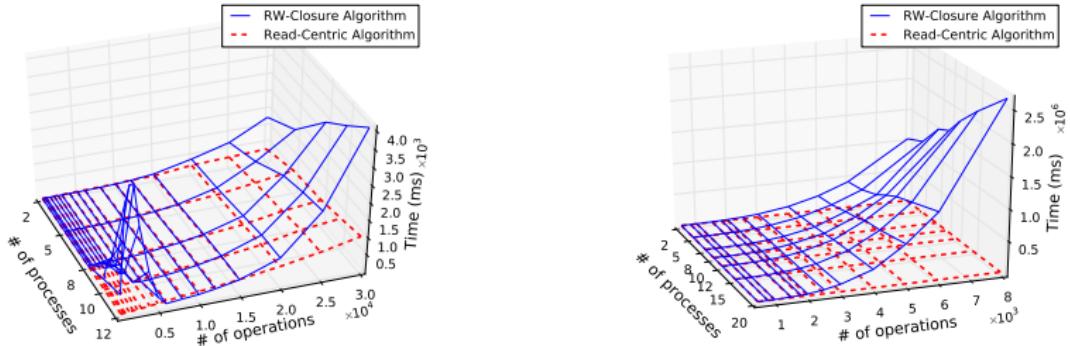
1. 考察 READ-CENTRIC 算法的实际效率 (*vs.* 漸近时间复杂度)
2. 对比 READ-CENTRIC 算法与 RW-CLOSURE 算法的效率

两类负载：

1. 随机生成的系统执行
2. 满足 PRAM 一致性的系统执行 ( $\approx$  最坏情况输入)

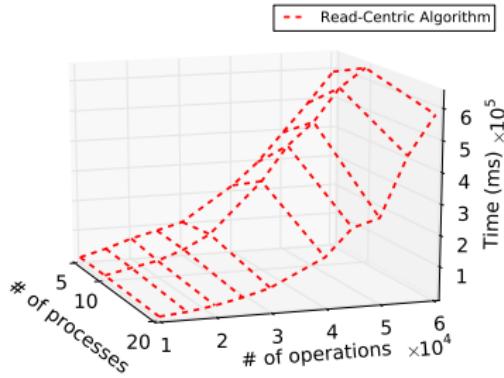
---

<sup>1</sup>机器配置: Intel Core i7 3.40GHZ, 4GB RAM.



RW-CLOSURE 算法与 READ-CENTRIC 算法在 (左) 随机生成的执行及 (右) 满足 PRAM 一致性的执行上的运行时间。

(右) 20 个进程、8,000 个操作：  
READ-CENTRIC 可获得 694 倍加速.



READ-CENTRIC 算法在满足 PRAM 一致性的执行上的运行时间

**READ-CENTRIC:** 20 个进程、60,000 个操作 < 600s <sup>1</sup>

**RW-CLOSURE:** 20 个进程、8,000 个操作 > 3,000s

<sup>1</sup> 用于测试，规模可用

$\mathcal{I}$ -Atomicity =  $i$  实时序 + 读写语义

$\mathcal{I}$ : Inversions

$$f(\{\text{inversions}\}) \leq i$$

定义框架

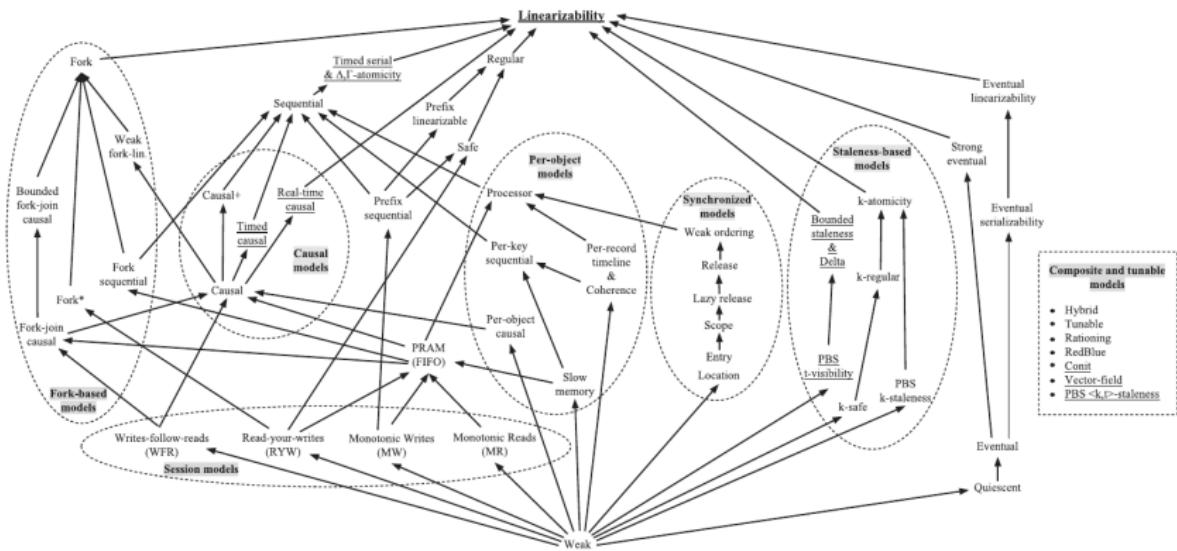


## 协议量化分析

[Lee and Welch, 2005] [Bailis et al., 2012] [Chatterjee and Golab, 2017]

PA2AM: Probabilistically-Atomic 2-Atomicity

# (50 种) 一致性模型 关系图 [Viotti and Vukolić, 2016] [Burckhardt, 2014]



建立统一的形式化框架