Principles and Practices for Foundational Operating System Kernel Verification

RIOS Group @ NII Japan

Ismail Kuru
Department of Computer Science
Drexel University

November 7, 2025

Outline

- Principles
  - Virtualization: Reasoning about Memory Address Virtualization [OOPSLA 2025]
  - Evolution: Reasoning against Changing Models [Under Submission]
  - Concurrency: A Type System for Read-Copy-Update Concurrency [ESOP 2019]
- Practice
  - Modal Reasoning Verification Patterns [PLOS 2025]
    - Chapter 0: Resource, Context, and Nominalization

Part I

Principles

Virtualization: A Case Study on Foundations for Memory Address Virtualization

The Essentials in Systems Programming

$$\underbrace{\text{pointer va}}_{\text{a virtual reference}} := \overbrace{\text{malloc (size)}}^{\text{a supposedly allocated physical resource}}$$

Memory Location Virtualization

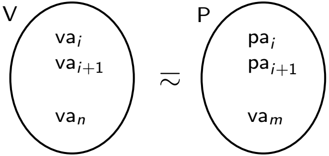


Figure: Virtualization: The Deception of Abundance

Memory Location Virtualization: Abstraction

An Address Space with Logical Name γ

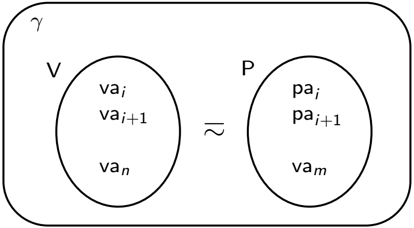


Figure: Address-Spaces: Named Containers for Virtual Memory Mappings

A Program Named γ_n

```
pointer va :=
malloc(size)
```

A Program Named γ_m

```
pointer va :=
malloc(size)
```

- A program is abstracted as a *named address-space*
- A container of *virtual-to-physical* memory resource mappings

Page Tables

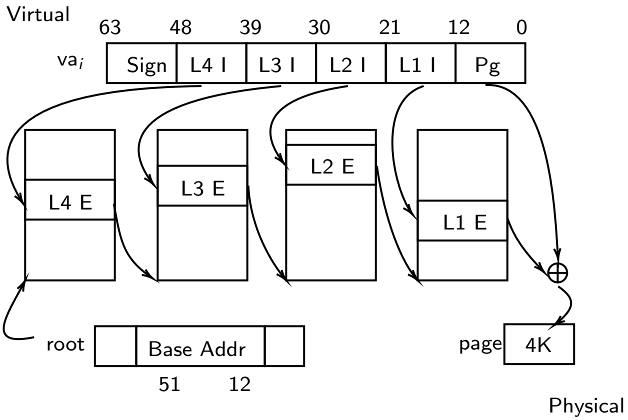


Figure: Page-Tables (**PT**): Data Structures for Address-Translation

A Complete Picture of Address-Space Abstraction

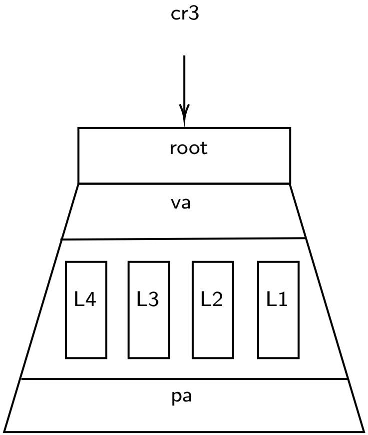


Figure: Depicting an Address-Space with its Essential Aspects

The Current View of Memory

The register cr_3 points to the current view of the memory, i.e., the loaded address space in the memory

.

Virtual Memory Management (VMM)

VMM as a General Resource Provider

”the virtual memory sub-system can be considered the core of a Solaris instance, and the implementation of Solaris virtual memory affects just about every other subsystem in the operating system” [McDougall and Mauro(2006)]

Memory Virtualization in TEE

- Protected enclaves within a process's address space - vTEEs
- Host OS swap pages in/out out of the TEE
- Data cannot be addressed by the principles in this talk

Sharing Physical Page Tables

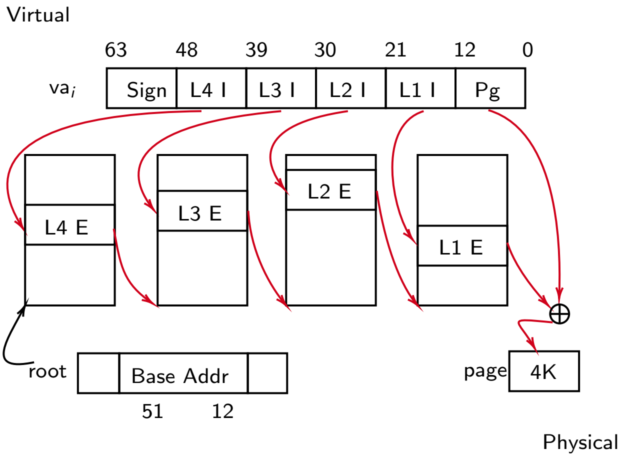
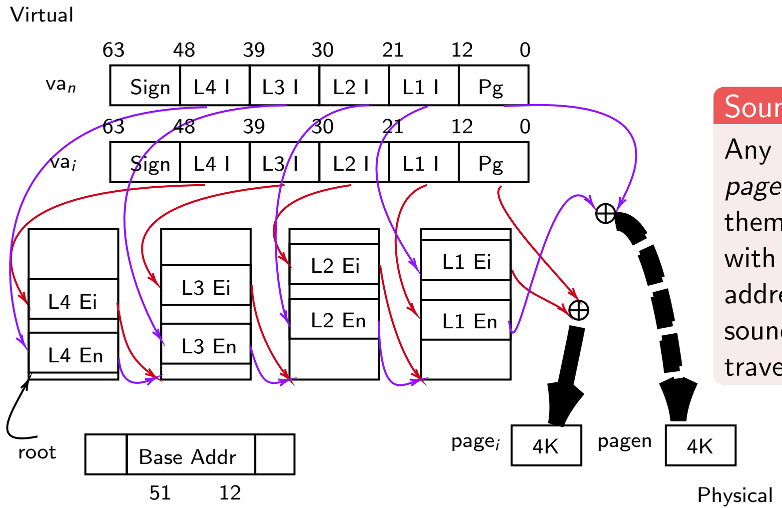


Figure: Accessing to the Page Referenced by L1 Entry

```
static pte_t *pte_nxt_table (pte_t *entry){
 pte_t *next;
 // If not already present, try to allocate
 if (!entry->present){
  if (!pte_alloc(&next)) {
   return NULL;
  }
  entry->pfn = PTE_PFN((uintptr_t) next);
  entry->present = 1;
  } else {
   uintptr_t next_phys_addr =
     PTE_PFN_TO_ADDR(entry->pfn);
   uintptr_t next_virt_addr = (uintptr_t)
     P2V(next_phys_addr);
   next = (pte_t *) next_virt_addr;
  }
  return next;
}
pte_t *walkpgdir(pte_t *l4, void *va){
 pte_t *l4_entry = &l4[L4I(va)];
 pte_t *l3 = pte_nxt_table(l4_entry);
 pte_t *l3_entry = &l3[L3I(va)];
 pte_t *l2 = pte_nxt_table(l3_entry);
 pte_t *l2_entry = &l2[L2I(va)];
 pte_t *l1 = pte_nxt_table(l2_entry);
 pte_t *l1_entry = &l1[L1I(va)];
 return l1_entry
}
```

Breaking Soundness in Sharing



Soundness of Traversal

Any update on the *shared page-tables*, which themselves are referenced with *physical memory* addresses, would break the soundness of any other traversal!

Managing Agnostic Memory Mappings

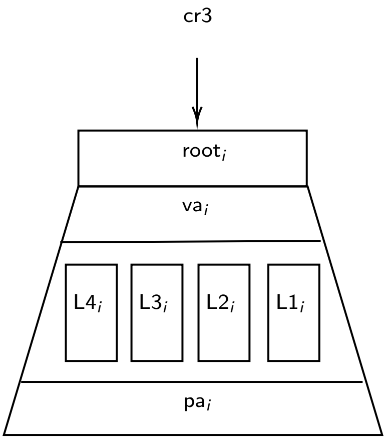


Figure: An Address Space with Unique Root Address $root_i$

Managing Agnostic Memory Mappings

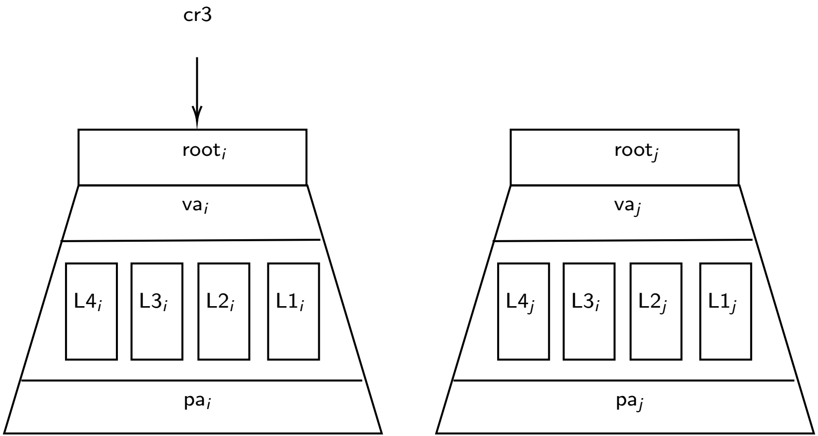


Figure: Two Address-Spaces with the Unique Root Addresses root_i and root_j

Managing Agnostic Memory Mappings

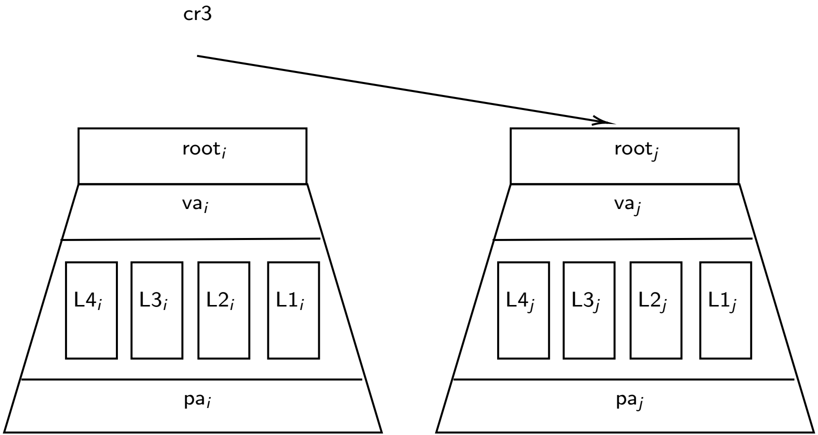


Figure: Switching Address-Spaces

Managing Agnostic Memory Mappings

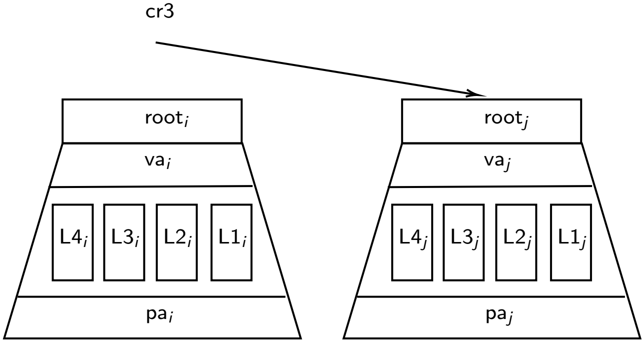


Figure: Switching Address-Spaces

Referring to Agnostic Resources

Unless we bookkeep to which address-space each of these virtual-to-physical mappings belongs, *which we never see in the practice of using virtual memory references*, we need to figure out a way of referring to these mappings as *they are only valid in their own address-spaces*.

Specifying Programs

$$\{P\}\ C\ \{Q\}$$

Separation Logic: Separating Conjuction

$$\textsc{Frame}\;\frac{\{P\}\; e\; \{Q\}}{\{P * R\}\; e\; \{Q * R\}}$$

Separation Logic: Ownership

- Well-known points-to assertion, e.g., memory_ref $\mapsto_q$ val
- Regarding the logical machinery, Iris **SL** enables encoding a generalized form ownership of *logical resources*
- A fragmental $\boxed{P}^{\gamma}$ ownership
  - Enabling coordinated access to logical resources
- Full $\boxed{P}^{\gamma}$ ownership
  - Enabling access to *update* logical resources, presented as *invariants*

Defining Some Ownersip Assertions

- Expected to have register ownership to be defined : $\mathsf{reg} \mapsto_r \mathsf{reg\_val}$
- Expected to have *physical memory* ownership defined: $\mathsf{pa} \mapsto_p \mathsf{val}$
- How about virtual memory references?

A Naive Attempt on Virtual-Pointsto

- Page and page table addresses are *physical*
- Purple (or red) path + bold black page references are *physical*
- Why don't we define *virtual* memory references in terms of the physical page-table and the final page references?

$$\mathrm{L_4\_L_1\_PointsTo(va,\ l4e,\ l3e,\ l2e,\ l1e,\ paddr)} + \mathit{paddr} \mapsto_p \mathrm{page\_val}$$

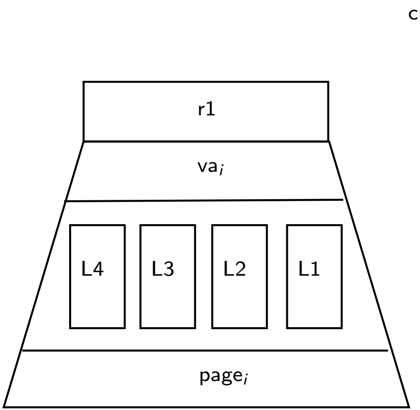
Tokens for Traversals

$$\underbrace{\mathsf{va} \hookrightarrow^{\delta}_{\mathsf{q}} \mathsf{pa}}_{\text{Ghost translation}} * \underbrace{\mathsf{pa} \mapsto_{\mathsf{p}} \{\mathsf{qfrac}\}\ \mathsf{val}}_{\text{Physical location}}$$

- Abstract the purple and red segment of page-table traversal into *logical summarization of the walk*
- Distribute the fragmental ownersip of the logical page-table summarization to virtual memory ownership

Habitat of Virtual Memory Mappings

$$\{[r1](va_i \mapsto page_i) * va_j \mapsto page_j\}cr3 := r1\{va_i \mapsto page_i * [r2](va_j \mapsto page_j)\}$$



Some Parts from Kernel Invariant

Definition (The Kernel Invariant for Page-Table Traversal with Virtual Page-Table Pointers)

$$\mathcal{I}\mathsf{ASpace}_{\mathsf{id}}(\theta, \Xi, m) \triangleq \mathsf{ASpace\_Lookup}_{\mathsf{id}}(\theta, \Xi, m) * \mathsf{GhostMap}(\mathsf{id}, \Xi) *$$
$$\left(\mathop{\Large\ast}_{(\mathsf{va},\mathsf{paddr}) \in \theta} \exists\, (\mathsf{l4e},\ \mathsf{l3e},\ \mathsf{l2e},\ \mathsf{l1e},\ \mathsf{paddr}).\, \mathsf{L_4\_L_1\_PointsTo}(\mathsf{va},\ \mathsf{l4e},\ \mathsf{l3e},\ \mathsf{l2e},\ \mathsf{l1e},\ \mathsf{paddr}) \right) *$$
$$\mathop{\Large\ast}_{(\mathsf{pa},\mathsf{level}) \in \Xi} \exists\, (\mathsf{qfrac},\ \mathsf{q},\ \mathsf{val},\mathsf{va}).\, \ulcorner \mathsf{va} = \mathsf{pa} + \mathsf{KERNBASE}\ \mathsf{level} > 1 \urcorner * \underbrace{\mathsf{va} \hookrightarrow^{\delta}_{\mathsf{q}} \mathsf{pa}}_{\text{Ghost translation}} * \underbrace{\mathsf{pa} \mapsto_{\mathsf{p}} \{\mathsf{qfrac}\}\ \mathsf{val}}_{\text{Physical location}} *$$
$$\underbrace{\ulcorner \mathsf{qfrac} = 1 \leftrightarrow \neg \mathsf{entry\_present}\ (\mathsf{val}) \urcorner}_{\text{Entry validity}} *$$
$$\underbrace{\left(\ulcorner \mathsf{present\_L}(\mathsf{val}, \mathsf{level}) \urcorner \mathrel{-\!\!*} \forall_{\mathsf{i} \in 0..511}.\, ((\mathsf{entry\_page}\ \mathsf{val}) + \mathsf{i} * 8) \hookrightarrow^{\mathsf{id}} \mathsf{level\text{-}1} \right)}_{\text{Indexing into next level of tables}}$$

where

$$\mathsf{present\_L}(\mathsf{val}, \mathsf{level}) \triangleq \mathsf{entry\_present}(\mathsf{val}) \wedge \mathsf{level} > 0$$

Specifying P2V

$\{P * \mathcal{I}\texttt{ASpace}_{\mathsf{id}}(\theta, \Xi \setminus \{\mathsf{entry}\}), m) * \mathsf{rbp\text{-}8} \mapsto_{\mathsf{v}} \mathsf{entry} * \texttt{rcx} \mapsto_{\mathsf{r}} \_ * \mathsf{entry} \mapsto_{\mathsf{id}} \_ * \mathsf{rtv} \hookrightarrow^{\delta s} \delta\}_{\mathsf{rtv}}$
$\{\mathsf{entry{+}KERNBASE} \mapsto_{\mathsf{vpte,qfrac}} (\mathsf{pte\_initialized}\ (\mathsf{entry\_val.pfn}))^{\urcorner}\}_{\mathsf{rtv}}$
$\{\texttt{rbp-16} \mapsto_{\mathsf{v}} (\mathsf{pte\_initialized}\ (\mathsf{entry\_val.pfn})) * \texttt{rax} \mapsto_{\mathsf{r}}\ \mathsf{table\_root}\ (\mathsf{pte\_initialize}(\mathsf{entry\_val.pfn}))\}_{\mathsf{rtv}}$
$\{\forall_{i \in 0\ \dots\ 511}.\ ((\mathsf{table\_root}\ (\mathsf{pte\_initialized}\ (\mathsf{entry\_val.pfn}))) + \mathsf{i} * 8) \hookrightarrow^{\mathsf{id}} \mathsf{v\text{-}1}\}$

```
;; uintptr_t next_virt_addr = (uintptr_t) P2V(entry.pfn<<12);
movabs KERNBASE, rcx
```

$\{\ldots * \texttt{rcx} \mapsto_{\mathsf{r}} \mathsf{KERNBASE} * \ldots\}_{\mathsf{rtv}}$

```
add     rcx, rax
```

$\{\ldots * \texttt{rax} \mapsto_{\mathsf{r}}\ \mathsf{table\_root}\ (\mathsf{pte\_initialize}(\mathsf{entry\_val.pfn})) + \mathsf{KERNBASE} * \ldots\}_{\mathsf{rtv}}$

```
...  ;;clean up the stack and return the rax value
```

Figure: Converting a physical address of a PTE to a virtual address (w/o instruction pointer or flag updates).

The Current Status of Machinery

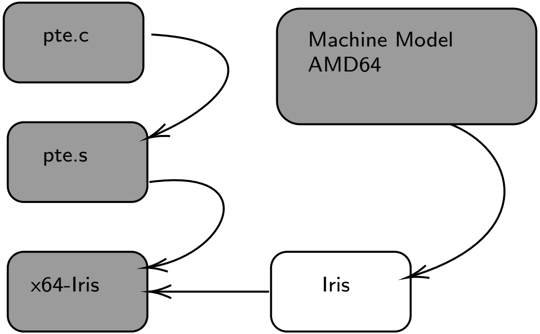


Figure: x64-Iris

- Dumping **.o** files
- Manuel treatment on **Xabs** instructions and field access

A Rough Quantification on the Current Status

Table: Line-of-Code Numbers for pte Verification

	C LoC A	Assembly LoC	Roqc Proof LoC
pte_get_next_table	12	45	3200
pte_walkpgdir	8	44	3200
pte_p2v	–	1	75
pte_switch_addrspace	–	18	350
pte_map_page	7	28	1750
pte_initialize	4	20	700

Table: Line-of-Code Numbers for x64-Iris Logic

	Roqc LoC
Soundness of Instructions Mentioned in the Presentation	50176 (The Complete Set of Instructions $\geq$ 1 Million)
VMM Related Logical Constructions	5554
Machine Model	6172

Evolution: Foundations for Specification Evolutionv & Protocol Reasoning

Protocols

- Interfaces are well-known abstractions in low-level systems
  - Device drivers, Virtual-File-Systems (VFS) etc.
- Protocols are well-know for specifying them

Protocols in TEE

- TEE Client API
- TEE Internal API
- Trusted Device Drivers

Specifying Protocols for Systems with STSes

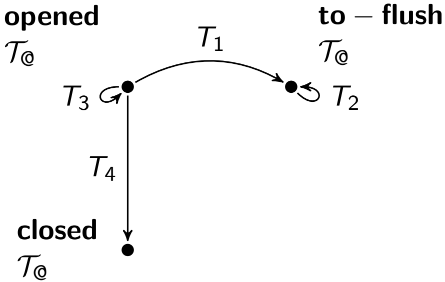


Figure: STS for Distributed File Protocol

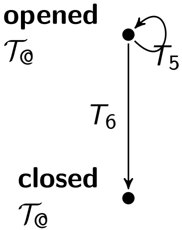


Figure: STS for Traditional File Protocol

Interacting with STSes

Modelling interactions of a client with a state machine via *token exchange*

Defining STSes

Definition (STS Definition following CaReSL's presentation [Turon et al.(2013)])

An STS π is given by:

1. a set of states $\mathcal{S}$,
2. a map from a state set of tokens $\mathcal{T} : \mathcal{S} \to \mathrm{TokSet}$,
3. a transition relation $\rightsquigarrow$ on states, which is then lifted to pairs of a state and token set:
$$(s; T) \rightsquigarrow (s'; T') \triangleq s \rightsquigarrow s' \wedge \mathcal{T}(s) \uplus T = \mathcal{T}(s') \uplus T'$$
4. an interpretation mapping states to state assertions $\varphi : \mathcal{S} \to \mathrm{Prop}$.

Propositional Kripke Model

Definition ((Propositional) Kripke Model [Hughes and Cresswell(1996)])

A Kripke model $\mathfrak{M}$ is a triple (W, R, V) where

- W is a set of "worlds"
- $R \subseteq W \times W$ is a relation called the *accessibility* relation between worlds
- $V : \mathrm{PropVar} \to \mathcal{P}(W)$ gives for each propositional variable p a set of worlds $V(p)$ where p is considered true

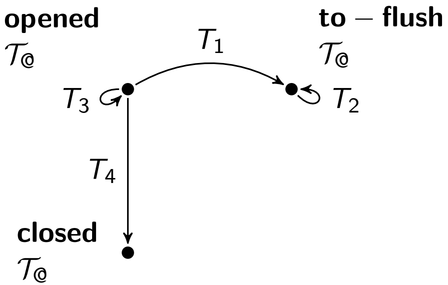
Bisimulations over Kripke Models

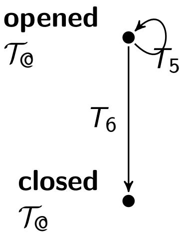
Definition ((Propositional) Bisimulation of Kripke Structures: $\mathfrak{M} \sim \mathfrak{M}'$.)

A *bisimulation* between (multimodal) Kripke structures $(W, R_{i\in I}, V)$ and $(W', R'_{i\in I}, V')$ is a relation $E \subseteq W \times W'$ satisfying:

- If $w\ E\ w'$, then w and w' satisfy the same propositional variables.
- If $w\ E\ w'$ and $w\ R\ v$, then there exists $v' \in W'$ such that $v\ E\ v'$ and $w'\ R'\ v'$
- If $w\ E\ w'$ and $w'\ R'\ v'$, then there exists $v \in W$ such that $v\ R\ v'$ and $w\ R\ v$

Intuition on Bisimulations over STSes





- More than just relating **STS**es in representation invariants per state.
- Bisimilar states can have different representation invariants.

Proof Indistinguishability

Knowing the proof of a client against the right (target STS conventionally π') *enables* deducing the proof against the bisimilar on the left (source STS conventionally π).

A Quick Tour on STS Assertions

- Invariants $\boxed{\varphi}^{\gamma}_{\pi}$, client capability $\boxed{s; \mathsf{T}}^{\gamma}$

STSAlloc

$$\frac{}{\varphi(s) \Rrightarrow \exists \gamma. \boxed{\varphi}^{\gamma}_{\pi} * \boxed{s; \mathsf{AllTokens} \setminus \mathcal{T}(s)}^{\gamma}}$$

STSOpen

$$\frac{}{\boxed{\varphi}^{\gamma}_{\pi} * \boxed{s; \mathsf{T}}^{\gamma} \Rrightarrow (\exists\, s'. \ulcorner(\mathsf{s0}, \mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi} (\mathsf{s}', \mathsf{T})\urcorner * \varphi(s) * \forall \mathsf{sl}',\ \mathsf{T}'. \ulcorner(\mathsf{s}', \mathsf{T}) \overset{\mathsf{guar.}^*}{\sqsubseteq}_{\pi} (\mathsf{sl}', \mathsf{T}')\urcorner * \varphi(\mathsf{sl}') \Rrightarrow \boxed{\mathsf{sl}; \mathsf{T}'}^{\gamma})}$$

UpdIsl

$$\frac{\alpha \text{ physically atomic} \qquad \forall s_0 .\ ((s; \mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi} (s_0; \mathsf{T})) \vdash \{\varphi(s_0) * P\}\ \alpha\ \{\exists s',\ \mathsf{T}'.\ (s_0; \mathsf{T}) \overset{\mathsf{guar}^*}{\sqsubseteq}_{\pi} (s'; \mathsf{T}') * \varphi(s') * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{s; \mathsf{T}}^{\gamma} * P\}\ \alpha\ \{\exists\, s', \mathsf{T}'. \boxed{s'; \mathsf{T}'}^{\gamma} * Q\}}$$

Figure: Iris STS Library [Jung et al.(2015)] simplified with later modality and invariant masks omitted

Decomposing Bisimilarity in STSes

The bisimulation $(\mathcal{M}(\pi, \pi', \varphi, \varphi', s, \mathsf{T}, \mathsf{U}))$ between two state machines, π and π' is composed of

- The source STS – π
- The target STS – π'
- The source STS's state interpretation function – φ
- The target STS's state interpretation function – φ'
- Token Embedding – $\epsilon_{\mathcal{S}} : \ \mathcal{S}(\pi) \mapsto \mathcal{S}(\pi')$
- State Embedding – $\epsilon_{\mathcal{T}} : \ \mathcal{T}(\pi) \mapsto \mathcal{T}(\pi')$
- The Law of Rely
- The Law of Guarantee
- The Law of Tolerance
- The state of source STS from which bisimulation is considered against any client interference with the token set T

Proof Translation

Obtain a proof rule utilizing the bisimulation to translate proofs between bisimilar state machines!

We Need This

$$\textsc{Bisim}\quad \frac{\pi \sim \pi' \qquad q \mathrel{\epsilon_{\mathcal{S}}} s \qquad q' \mathrel{\epsilon_{\mathcal{S}}} s' \qquad \{\boxed{s; \epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s'; \mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q; \mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q'; \mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

We Use This

UpdIsl

$$\frac{\alpha \text{ physically atomic} \qquad \forall s_0 \,.\, ((s;\mathsf{T}) \overset{\mathsf{rely}^*}{\sqsubseteq}_{\pi} (s_0;\mathsf{T})) \vdash \{\varphi(s_0) * P\}\ \alpha\ \{\exists s',\ \mathsf{T}' \,.\, (s_0;\mathsf{T}) \overset{\mathsf{guar}^*}{\sqsubseteq}_{\pi} (s';\mathsf{T}') * \varphi(s') * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{s;\mathsf{T}}^{\gamma} * P\}\ \alpha\ \{\exists\, s', \mathsf{T}' . \boxed{s';\mathsf{T}'}^{\gamma} * Q\}}$$

Bisim

$$\frac{\pi \sim \pi' \qquad q \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s;\epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ C\ \{\boxed{s';\mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi}^{\gamma}_{\pi} \vdash \{\boxed{q;\mathsf{T}}^{\gamma}_{\pi} * P\}\ C\ \{\boxed{q';\mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

Invariants of File Protocols

Definition (File Protocol Invariants)

$$\varphi_{\text{distributedfile}}\ (\ \ell\ ,\ \mathsf{R}\ )(\ s\ )\ \triangleq \left\{ \begin{array}{ll} \text{match s with} & \\ \text{to} - \text{flush} \Rightarrow & \mathsf{R} * \exists\ \text{fs. isValidDirty(fs)}* \\ & \quad \ell \mapsto (\text{fs.}\mathit{id}, \text{fs.status} = \text{dirty}) \\ \quad \text{opened} \Rightarrow & \quad \mathsf{R} * \exists\ \text{fs. isValid(fs)}* \\ & \quad \ell \mapsto (\text{fs.}\mathit{id}, \text{fs.status} = \text{clean}) \\ \quad \text{closed} \Rightarrow & \quad \exists\ \text{fs. isValidClosed(fs)}* \\ & \quad \ell \mapsto (\text{fs.}\mathit{id}, \text{fs.status} = \text{closed}) \end{array} \right\}$$

$$\varphi_{\text{file}}\ (\ \ell\ \mathsf{R}\ )(\ s\ )\ \triangleq \left\{ \begin{array}{ll} \text{match s with} & \\ \quad \text{opened} \Rightarrow & \begin{array}{l} \mathsf{R} * \exists\ \text{fs. isValid(fs)}* \\ \ell \mapsto (\text{fs.}\mathit{id}, \text{fs.status} = \text{clean} \vee \text{dirty}) \end{array} \\ \quad \text{closed} \Rightarrow & \begin{array}{l} \quad \exists\ \text{fs. isValidClosed(fs)}* \\ \quad \ell \mapsto (\text{fs.}\mathit{id}, \text{fs.status} = \text{closed}) \end{array} \end{array} \right\}$$

Keeping Promises

$$\textsc{Transfer File Write}\quad \dfrac{\pi \sim \pi' \qquad \text{opened} \in_{\mathcal{S}} s \qquad q' \in_{\mathcal{S}} s' \qquad \{\boxed{s; \epsilon_{\overline{\mathcal{T}}}(\mathcal{T})}^{\gamma}_{\pi'} * P\}\ \text{write}\ \ell\ \text{new\_val}\ \{\boxed{s'; \mathsf{T}'}^{\gamma}_{\pi'} * Q\}}{\boxed{\varphi_{\text{distributedfile}}}^{\gamma}_{\pi} \vdash \{\boxed{\text{opened}; \mathsf{T}}^{\gamma}_{\pi} * P\}\ \text{write}\ \ell\ \text{new\_val}\ \{\boxed{q'; \mathsf{T}'}^{\gamma}_{\pi} * Q\}}$$

The Law of Rely

Theorem (The Law of Rely)

$$\forall s'.(s; T) \overset{\text{rely}^*}{\sqsubseteq}_{\pi} (s'; T) \leftrightarrow$$
$$(\forall_{s_1, s_1', T_1}. \epsilon_{\mathcal{S}}(s, s_1) \rightarrow \epsilon_{\mathcal{S}}(s', s_1') \rightarrow \epsilon_{\mathcal{T}}(T, T_1) \rightarrow (s_1; T_1) \overset{\text{rely}^*}{\sqsubseteq}_{\pi'} (s_1'; T_1)$$

- We do not drop any client interference with capabilities T
- Indetification of the states that are tolerant to the client interference from which the STS can take steps (Guarantee)
- Bookkeeping of the client interference needed!
- Identifying the valid *pre* state

The Law of Guarantee

Theorem (Guarantee Bisim without Invariants)

$$
\begin{aligned}
&\forall_{q',q,T'}.\, \epsilon_{\overline{\mathcal{T}}}(T) \equiv T' \rightarrow \epsilon_{\mathcal{S}}(s,q) \rightarrow (q;\, T') \overset{\text{rely}^*}{\sqsubseteq}_{\pi'} (q';\, T') \rightarrow \\
&\qquad \forall_{q'',T''}.\, (q';\, T') \overset{\text{guar.}}{\sqsubseteq}_{\pi'} (q'';\, T'') \rightarrow \\
&\qquad\qquad \exists_{s',s'',T'_0,T''_0}.\, (s';\, T'_0) \overset{\text{guar.}}{\sqsubseteq}_{\pi} (s'';\, T''_0) \wedge \\
&\qquad\qquad\qquad \epsilon_{\mathcal{S}}(s') = q' \wedge \epsilon_{\mathcal{S}}(s'') = q'' \wedge \epsilon_{\overline{\mathcal{T}}}(T'_0) \equiv T' \wedge \epsilon_{\overline{\mathcal{T}}}(T''_0) \equiv T''
\end{aligned}
$$

- *Under the embedded client interference*, the steps taken by the target STS must be countered by a one in the source STS
- From target STS to source STS
- Identifying the valid *post* state

Soundness

Theorem (Soundness)

The updated abstract state from UPDISL *is preserved by the bisimulation.*

Ongoing Work

- Homomorphisms for general form of specifications (i.e., more than STSes)
- Exploit another obvious application fields, e.g., device drivers
- Only Iris pluggable?

Concurrency: Semantic Type Assertions for Deferred Memory Reclamation Schemes

What is Deferred Memory Reclamation?

- Both *reader* and *writer* threads accessing to a memory location *simultaneously*
- The write *waits* the readers that are already on the same memory location — i.e., *Grace Period*
- After the grace period end, the grace period ends, i.e, the readers leave the memory location, it is safe to *reclaim* the memory location
- Different schemes: Hazard Pointers (Maged M. Michael 2004), Read-Copy-Update (PE McKenney and JD Slingwine [McKenney and Slingwine(1998)])

RCU Semantics

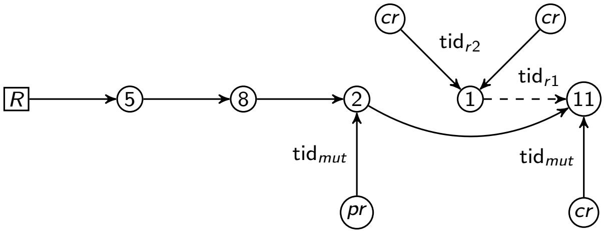
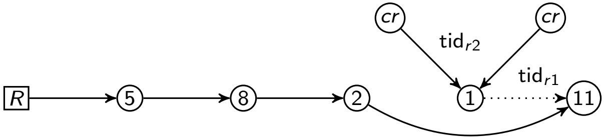


Figure: tid$_{mut}$ unlinks the node with value 1.



to-free list

...	(F[s(1,tid$_{mut}$) $\rightharpoonup$ tid$_{r1}$, tid$_{r2}$]	···	

RCU Semantics

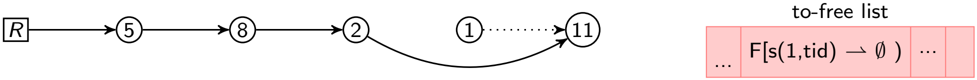


Figure: Bounding threads, tid_{r1} and tid_{r2} exit ReadBlock.

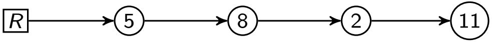


Figure: Reclaimed the node 1

Type Assertions for RCU

```
struct BagNode{
  int data;
  BagNode<rcuItr> Next;
}
BagNode<rcuRoot> head;
```

```
void add(int toAdd){
WriteBegin;
BagNode nw = new;
{nw: rcuFresh{}}
nw.data = toAdd;
{head: rcuRoot, par: undef, cur: undef}
BagNode<rcuItr> par,cur = head;
{head: rcuRoot, par: rcuItrϵ{}}
{cur: rcuItrϵ{}}
cur = par.Next;
{cur: rcuItr Next{}}
{par: rcuItrϵ{Next ↦ cur}}
while(cur.Next != null){
  {cur: rcuItr(Next)^k.Next{}}
  {par: rcuItr(Next)^k{Next ↦ cur}}
  par = cur;
  cur = par.Next;
}
...
WriteEnd;
}
```

A Taste of Soundness on Type System for RCU

- Soundness on top of Views Framework (Dinsdale-Young et.al. [**?**])
  - Logical state with its observation-map , free-list etc.
  - Denotation of types encoding the post-environment of any type accurately

$$[\![\Gamma, x : \mathsf{rcultr}\, \rho\, \mathcal{N}[y : \mathsf{rcultr}]]\!]_{M,tid}$$

- Global Invariants
  - Unlinked Reachability:
  - Delayed Ownership Transfer and Reader in Freelist:
- Discharging these invariants once as a part of soundness
  - No need to prove them for each different client

Remarks

- Simpler that full-blown program logics: Tassarotti et al. (PLDI 2015) [Tassarotti et al.(2015)], Fu et.al., Gotsman et.al.(ESOP 2013)
- The first general operational model for RCU-based memory management
- Based on our suitable abstractions for RCU in the operational semantics
  - Decoupling the memory-safety proofs from the underlying reclamation model
  - Similar is done for correctness by Meyer and Wolff (POPL 2019)
- Applicability/Usability
  - The first safety proof RCU client Citrus Binary Search Tree (Maya Arbel and Hagit Attiya PODC 2014)
  - Linked-list based bag implementation (McKenney Technical Report 2015)
- More type rules in the paper
  - Refinement rules for control flows
  - A simple type system for readers
  - Entering and exiting read/write-side critical sections

Future Directions

- Deploying it as Clang front-end
  - Abstract operational semantics can handle "classical RCU"
  - But optimized "batch lists" in Linux kernel? Refinement with our abstract model?
- Rust ownership
  - When published Rust's ownership was not able to handle RCU-like programming pattern
  - Now there is a set of RCU types
- Go adopted similar pattern in the existence of garbage-collector
  - Captured by our operational semantics
  - Async-free + Free list
- Beyond memory-safety? Tolerance to *stale data*

Part II

Practice

Modal Verification Patterns for Low-Level Systems

Practice of Program Logic Design for Low-Level Systems

- What is the conceptualized thinking in designing program logic for a low-level system?
- Can we identify certain patterns?

The Essentials in Systems Programming

$$\underbrace{\text{FILE}*\ \text{fptr}}_{\text{a file handle}} := \overbrace{\text{fopen(filename, mode);}}^{\left\{\begin{array}{c}\text{a supposedly accesible data at somewhere in the computer} \\ \text{which makes its potential mode unknown : in\_memory or on disk or ...}\end{array}\right\}}$$

File Page Virtualization

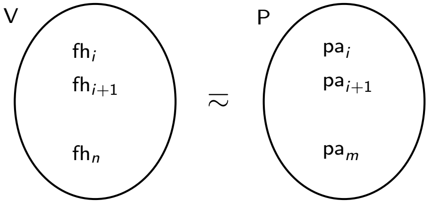
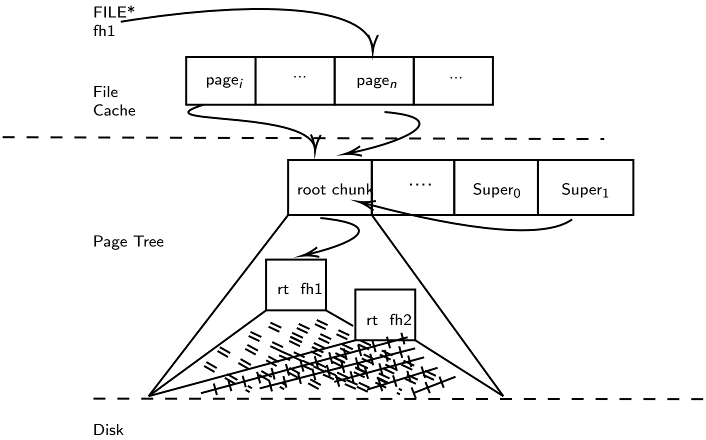


Figure: Virtualization: The Deception of Disk-Page Abundance

A Global Disk-Page Tree



File Data Virtualization: Wait! Maybe a Bit More!

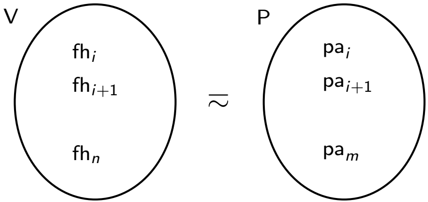


Figure: Virtualization: The Deception of Disk-Page Abundance Parameterized *under* **Some Consistency Model**

File Data Virtualization: Abstraction

A Disk-Page Tree with Logical Name γ_n?

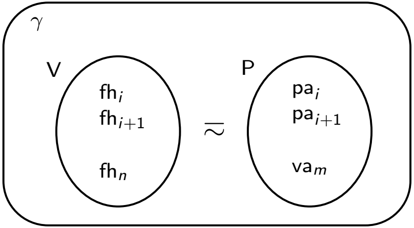


Figure: A Global Disk-Page Tree: Named Containers for File-Page to Disk-Page Mappings?

An Updated File in the Named Disk-Tree γ_n

```
FILE* fh1 :=
  write(data)
```

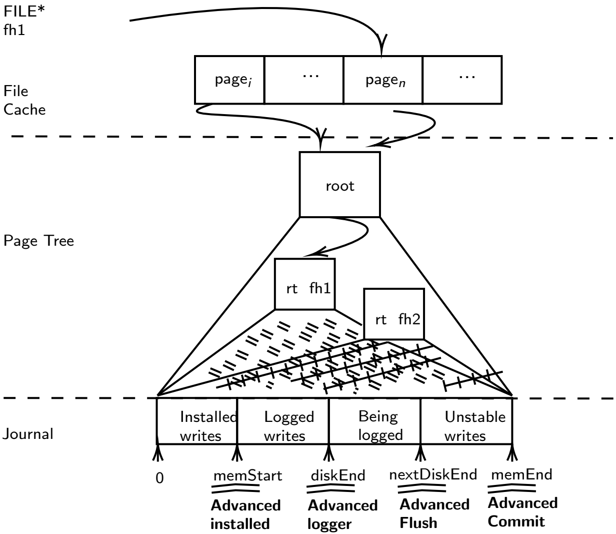
An Read-Only Access to a File in the Named Disk-Tree γ_n

```
FILE fh2 :=
  read(data, sz)
```

- Could a global disk tree as a container work for virtual-to-physical disk resources?
- Maybe? But not always!

A Global File Page Tree with Multiple Views

Consistency models can impose multi-mode-views on the disk page tree



An Example for a Consistency Model: Journalling

- Indices uniquely naming the consistent pieces of disk and the updates to be inserted into the disk
- Certain pages of the global tree are valid under different *views* to it
- Recovery, Atomicity ...

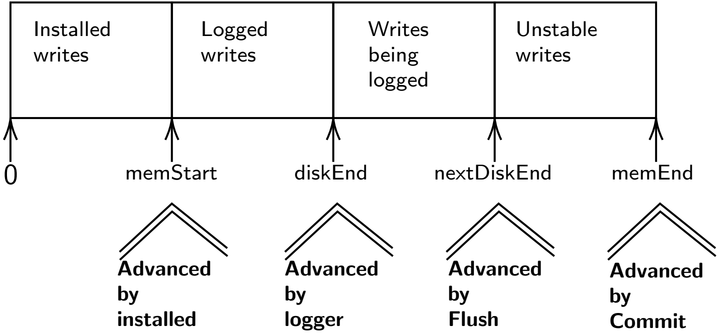


Figure: Depicting Journalling Model

Another Example for a Consistency Model: Copy-On-Writes Filesystems

- Updates are done on newly allocated resources
- Snapshots are collections of updates
- A uniquely identifying snapshotting identifier naming the consistent pieces of disk.
- Snapshot updates appear on the disk atomically: always have a consistent *view* of the disk-tree
- Recovery, Atomicity ....

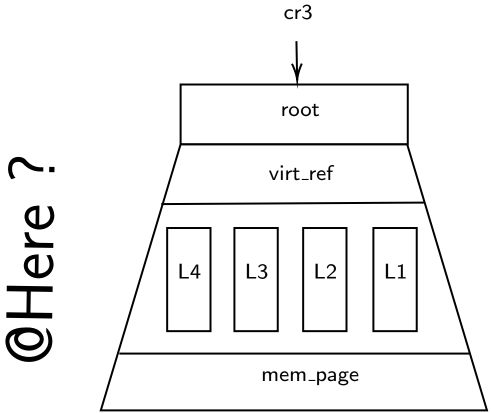
Resource: Unital Element as a Fact Matters Most

$\{P\}\ C\ \{Q\}$

- What matters most inside P for for the program action C *contingently*?
- Well-known points-to assertion, e.g., virt_ref $\mapsto$ mem_page

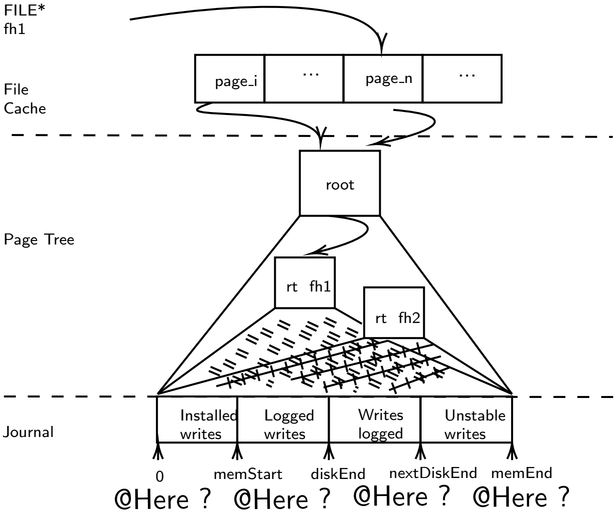
A Virtual Memory Pointsto

- virt_ref $\mapsto$ mem_page



A Disk Page Pointsto

- An expected points-to assertion, e.g., page_ref $\mapsto_q$ page

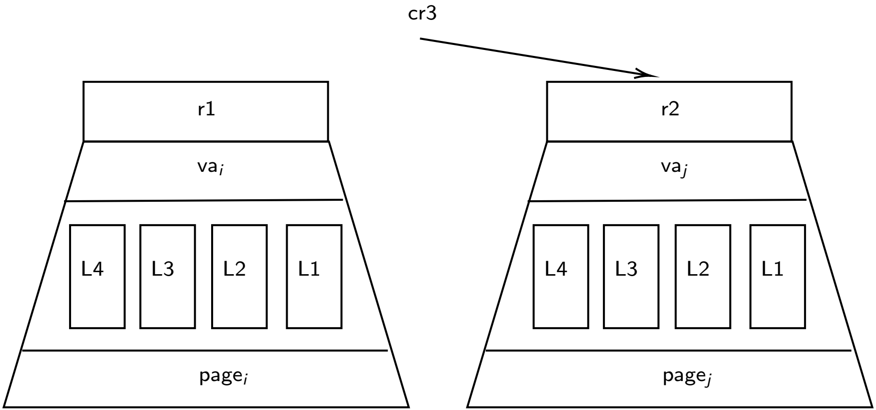


@Here ?: Resource Context

- The *habitat* of a resource determining its scope of validity

Habitat of Virtual Memory Mappings

$$\{[r1](va_i \mapsto page_i) * va_j \mapsto page_j\} cr3 := r1 \{va_i \mapsto page_i * [r2](va_j \mapsto page_j)\}$$



Habitats for Disk Resources in Journaling

- A specification of *recovery* would require both
  - Explicitly naming on the resources that can be inferred from their uniquely identifying resource context name
  - Losing duality of resource contexts in specifications

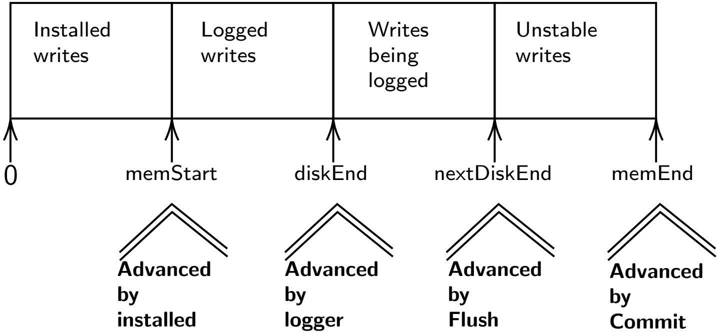


Figure: Depicting Journalling Model

Modal Decomposition of Program-Logics

Modality	Context	Elements	Nominalization	Context Steps
Post-Crash+	$\Diamond\ P$	$\ell \mapsto_n^{\overline{\gamma}} v$	Strong	Crash Recovery
NextGen!	$\overset{t}{\hookrightarrow}\ P$	Own (t(a))	Strong	Determined Based on the Model*
StackRegion*	$\overset{ICut^n}{\hookrightarrow}\ P$	$\boxed{\mathsf{n}}\ \ell \mapsto v$	Strong	Alloc and Return to/from stack
Actor#	$@_\iota$ P	Variable values	Weak	Send Message
Memory-Fence^x	$\triangle_\pi$ and $\triangledown_\pi$	$\ell \mapsto v$	Weak	Fence Acquire and Release
Address Space?	[r]P	$\ell \mapsto v$	Weak	Address Space Switch
Ref-Count&	$@_\ell$ P	$\ell_1 \mapsto v$	Weak	Allocating, Dropping and Sharing a Reference

$*$*The StackRegion Modality is an instance of NextGen (called the Independence Modality in* [Vindum et al.(2025)]).
+[Chajed(2022), Chajed et al.(2019), Tej Chajed and contributors(2023)]
! [Vindum et al.(2025)]
[Gordon(2019)]
? [Kuru and Gordon(2024), **?**]
& [Wagner et al.(2024)]
x [Doko and Vafeiadis(2016), Doko and Vafeiadis(2017), Dang et al.(2019)]

Remarks

- This paper:
  - First steps in identifying key pieces in building a program logic for real systems
  - Nominalization as "*naming resource contexts and its resources*" is in the paper
- The verification pattern concepts are not specific to separation logic!
  - Actor modelling in Dafny [Gordon(2019)]
- This is an introductory chapter
  - The next chapter is on the interaction between resource contexts.

Tej Chajed. 2022.
Verifying a concurrent, crash-safe file system with sequential reasoning.
Ph.D. Dissertation. Machetutes Institute of Technology, Cambridge, MA.
Available at https://dspace.mit.edu/handle/1721.1/144578.

Tej Chajed, Joseph Tassarotti, M. Frans Kaashoek, and Nickolai Zeldovich. 2019.
Verifying concurrent, crash-safe systems with Perennial. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 243–258.
https://doi.org/10.1145/3341301.3359632

Hoang-Hai Dang, Jacques-Henri Jourdan, Jan-Oliver Kaiser, and Derek Dreyer. 2019.
RustBelt meets relaxed memory.
Proc. ACM Program. Lang. 4, POPL, Article 34 (Dec. 2019), 29 pages.
https://doi.org/10.1145/3371102

Marko Doko and Viktor Vafeiadis. 2016.
A Program Logic for C11 Memory Fences. In *Proceedings of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 9583* (St. Petersburg, FL, USA) *(VMCAI 2016)*. Springer-Verlag, Berlin, Heidelberg, 413–430.
https://doi.org/10.1007/978-3-662-49122-5_20

Marko Doko and Viktor Vafeiadis. 2017.
Tackling Real-Life Relaxed Concurrency with FSL++. In *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on*

Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings (Uppsala, Sweden). Springer-Verlag, Berlin, Heidelberg, 448–475.
https://doi.org/10.1007/978-3-662-54434-1_17

Colin S Gordon. 2019.
Modal assertions for actor correctness. In *Proceedings of the 9th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*. 11–20.

George Edward Hughes and Max J Cresswell. 1996.
A new introduction to modal logic.

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015.
Iris: Monoids and Invariants As an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Mumbai, India) *(POPL '15)*. ACM, New York, NY, USA, 637–650.
https://doi.org/10.1145/2676726.2676980

Ismail Kuru and Colin S. Gordon. 2024.
Modal Abstractions for Virtualizing Memory Addresses.
arXiv:2307.14471 [cs.PL] https://arxiv.org/abs/2307.14471

Richard McDougall and Jim Mauro. 2006.
Solaris internals: Solaris 10 and OpenSolaris kernel architecture.
Pearson Education.

Paul E McKenney and John D Slingwine. 1998.

Read-copy update: Using execution history to solve concurrency problems. In *Parallel and Distributed Computing and Systems*. 509–518.

Joseph Tassarotti, Derek Dreyer, and Viktor Vafeiadis. 2015.
Verifying Read-copy-update in a Logic for Weak Memory. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI 2015)*. ACM, New York, NY, USA, 110–120.
https://doi.org/10.1145/2737924.2737992

Josep Tassarotti Tej Chajed and contributors. 2023.
Post-crash modality in Perennial's Coq Mechanization.
https://github.com/mit-pdos/perennial/blob/master/src/goose_lang/crash_modality.v

Aaron Turon, Derek Dreyer, and Lars Birkedal. 2013.
Unifying Refinement and Hoare-Style Reasoning in a Logic for Higher-Order Concurrency. In *ICFP*.

Simon Friis Vindum, Aïna Linn Georges, and Lars Birkedal. 2025.
The Nextgen Modality: A Modality for Non-Frame-Preserving Updates in Separation Logic. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Denver, CO, USA) *(CPP '25)*. Association for Computing Machinery, New York, NY, USA, 83–97.
https://doi.org/10.1145/3703595.3705876

Andrew Wagner, Zachary Eisbach, and Amal Ahmed. 2024.
Realistic Realizability: Specifying ABIs You Can Count On.
Proc. ACM Program. Lang. 8, OOPSLA2, Article 315 (Oct. 2024), 30 pages.
https://doi.org/10.1145/3689755