Modeling and Abstraction of Memory Management in a Hypervisor

Pauline Bolignano [1,2] Thomas Jensen [1] Vincent Siles [2]

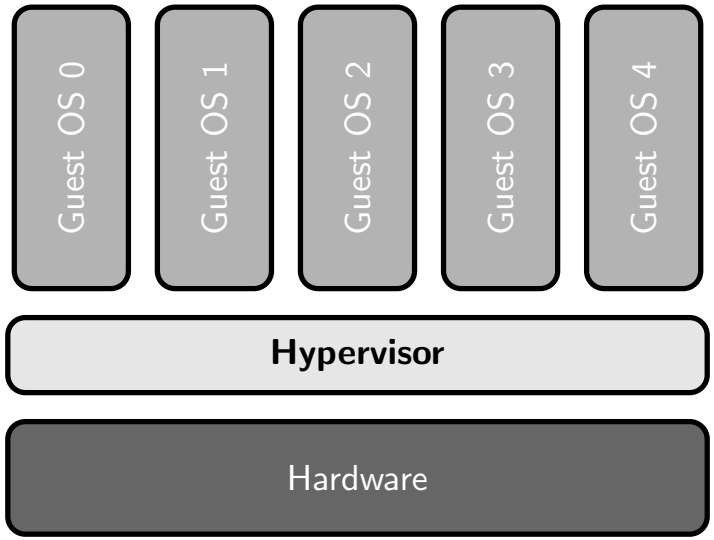
[1]Inria, Rennes, France

[2]Prove & Run, Paris, France

April 6th, 2016

Hypervisors

- Allow to run several OSes on the same platform.
- Allow to run legacy OS.
- Are security critical components.
- Bugs are severe.
  - Xen bug in Page Tables Management [1].



[1] https://github.com/QubesOS/qubes-secpack/blob/master/QSBs/qsb-022-2015.txt

Targeted Properties

Isolation:

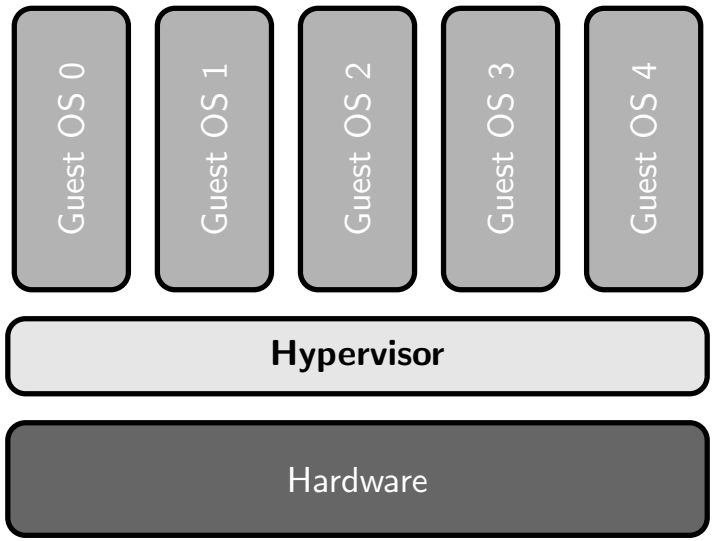
- Integrity.
- Confidentiality.

On:

- Memory.
- Registers.

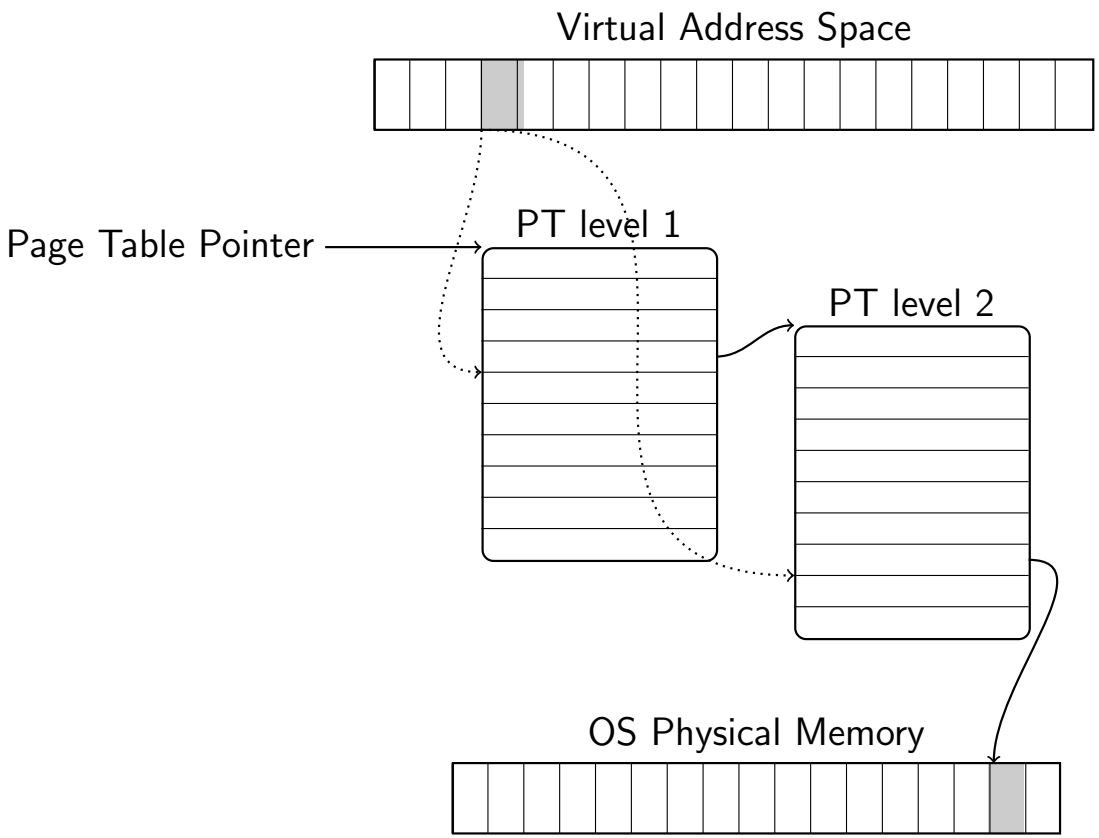
Out of the model:

- Side Channels.



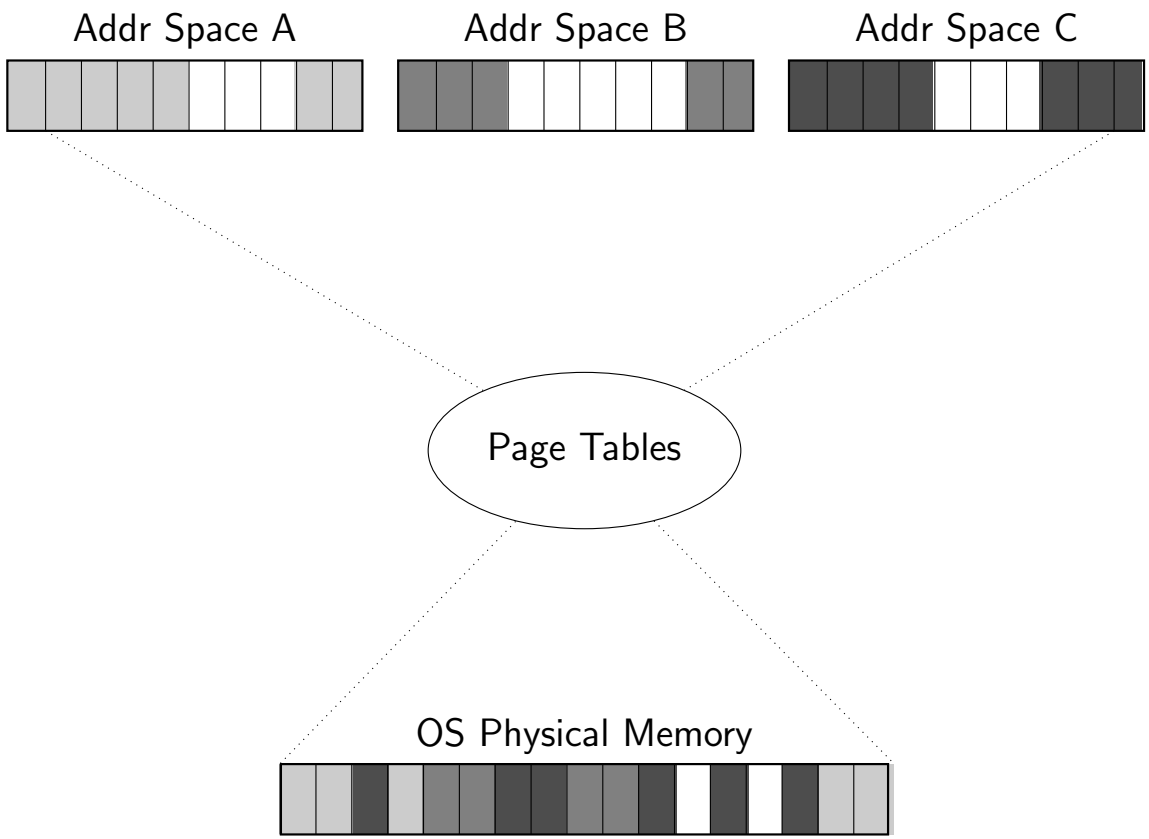
Memory Management for an OS

- The MMU translates virtual addresses to physical addresses with the Page Tables.

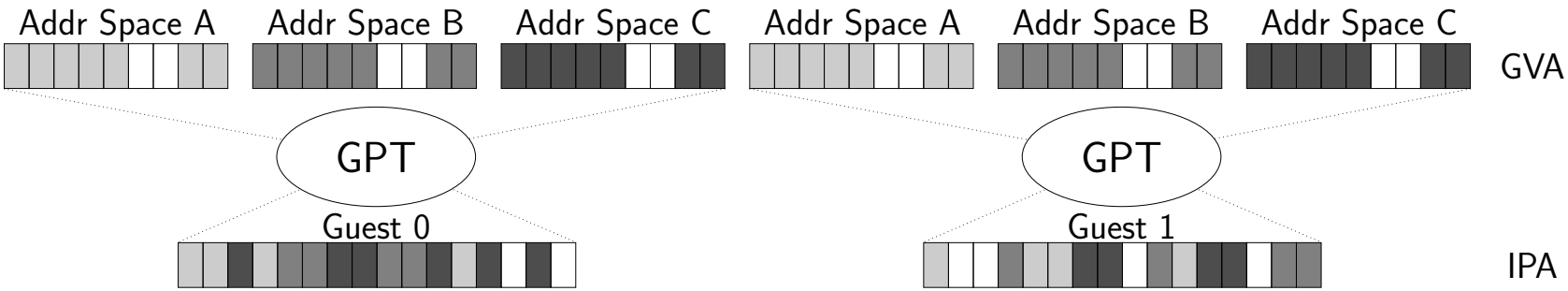


Memory Management for an OS

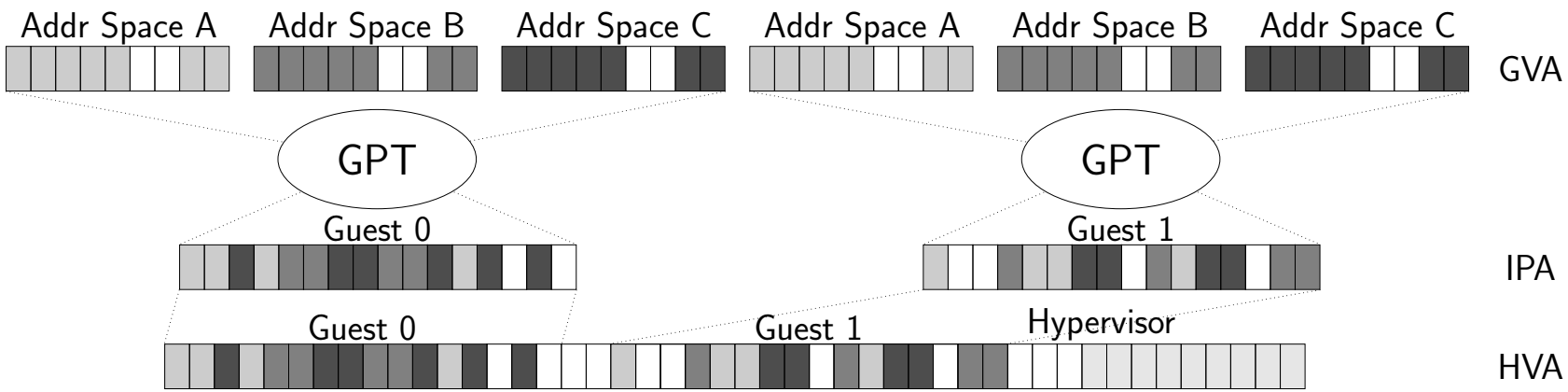
- The MMU translates virtual addresses to physical addresses with the Page Tables.
- Each Page Table defines an address space.



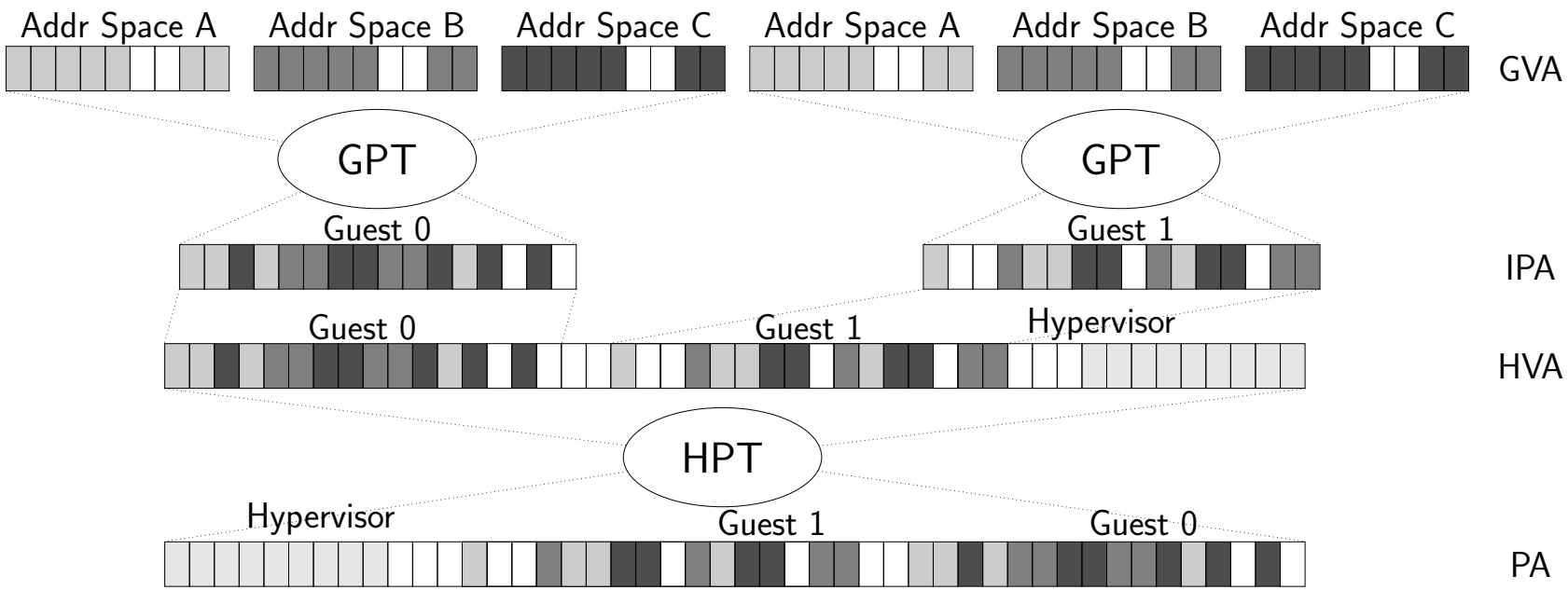
Memory Management for a Hypervisor



Memory Management for a Hypervisor

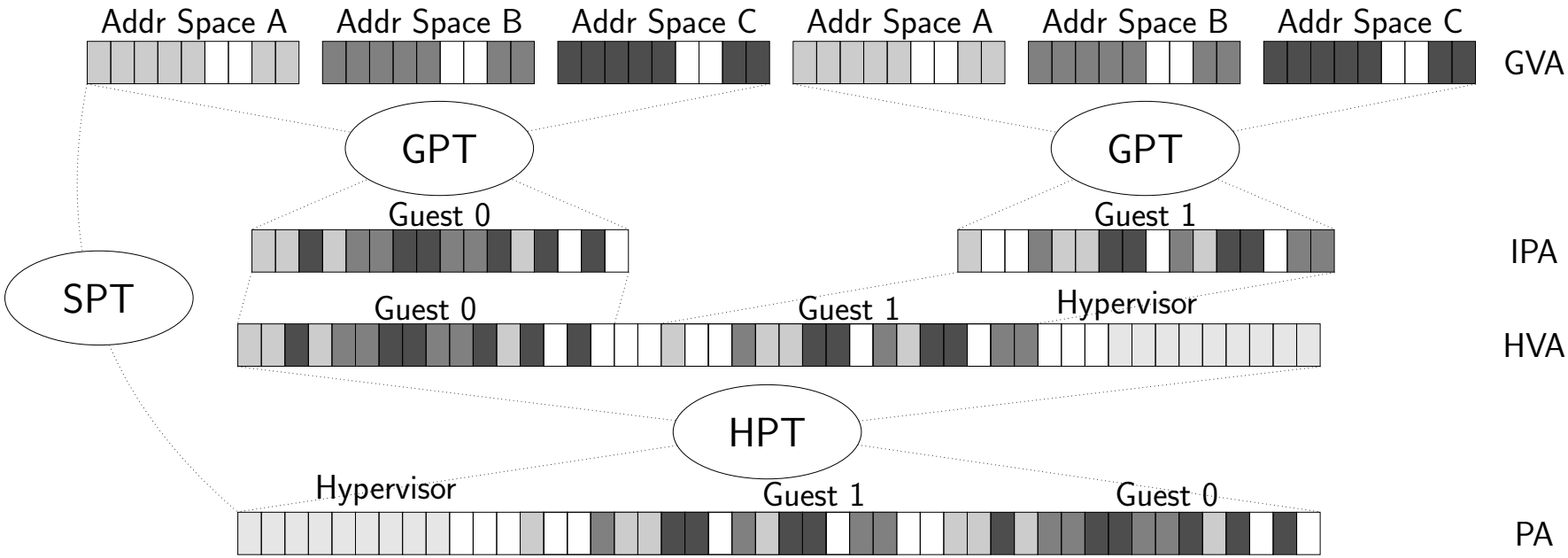


Memory Management for a Hypervisor



Memory Management for a Hypervisor

Shadow Page Tables combine Guest Page Tables and Hypervisor Page Tables.



Context

Prove & Run:

- Develops Tools:
  - Language (Smart).
  - Prover.
- Develops and proves highly secure systems:
  - Micro-kernels.
  - Hypervisors.

Our Work

Which hypervisor?

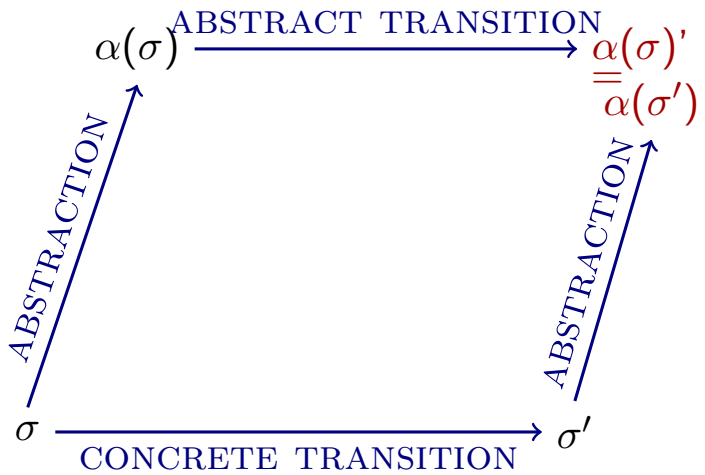
- Developed by SecT team (TU Berlin).
- Para-virtualized.
- With Shadow Page Tables.
- With static configuration of memory access rights.

Which tools?

- Prove & Run Language and Prover.

Outline

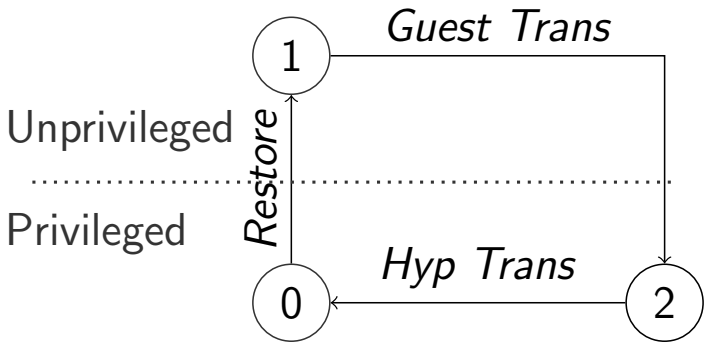
Proof by Abstraction



- Isolation on the **abstract** system implies isolation on the **concrete** system.
- The abstract state must be **precise** enough to **express our properties** on it.

Correspondence between Concrete and Abstract Transitions

Flow:



Hypervisor Transitions:

Concrete	Abstract
Switch Page Fault with MMU Page Fault without MMU Flush Activate MMU Deactivate MMU	Memory Management
Schedule	Schedule
Inject SWI Inject UND Inject ABT Access Privileged Registers	Modify Registers
Handle IRQ Passthrough IRQ Fetch IRQ	Nop

Abstract State

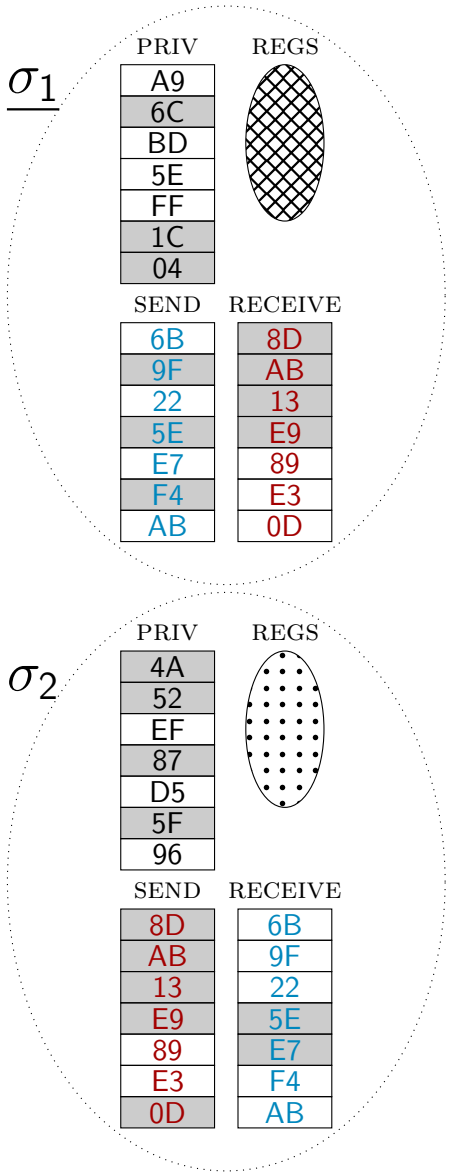
Abstract State :

$$\sigma_\alpha = \langle curr, \sigma_1, ...\sigma_n \rangle$$

Where, for all i :

$$\sigma_i = \langle regs, priv, \langle s_1, ..., s_n \rangle, \langle r_1, ..., r_n \rangle \rangle$$

- $priv$: private segment of i.
- $\sigma_i.s_k$: send segment from i to k.
- $\sigma_i.r_k$: receive segment from k to i, in particular $\sigma_i.r_k = \sigma_k.s_i$.



Properties on the Abstract Model

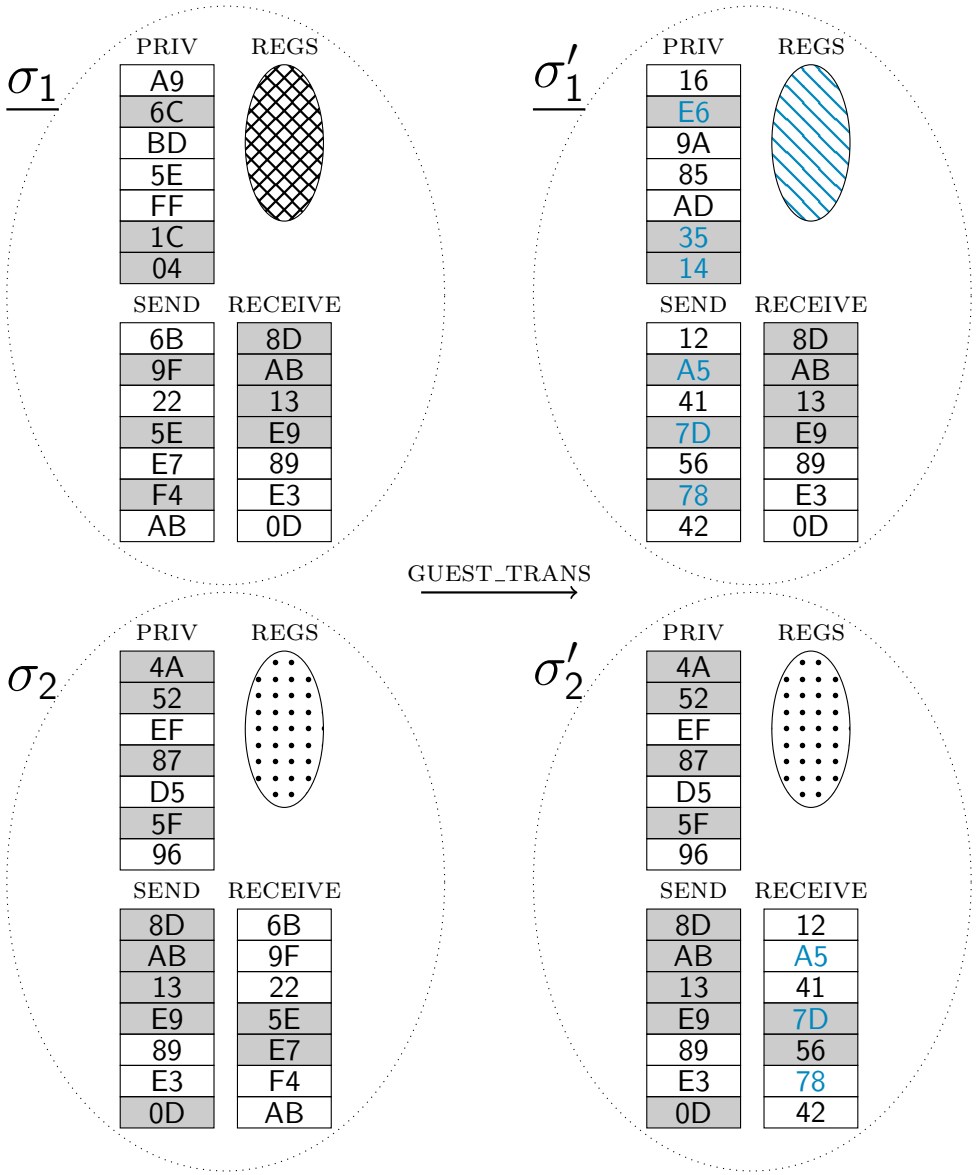
Integrity Theorem: Let i and j be two guest indexes such that $i \neq j$. Consider a transition where j is the running guest. If

$$\langle j, \sigma_1, ..., \sigma_i, ..., \sigma_n \rangle \rightarrow \langle j', \sigma'_1, ..., \sigma'_i, ..., \sigma'_n \rangle$$

then

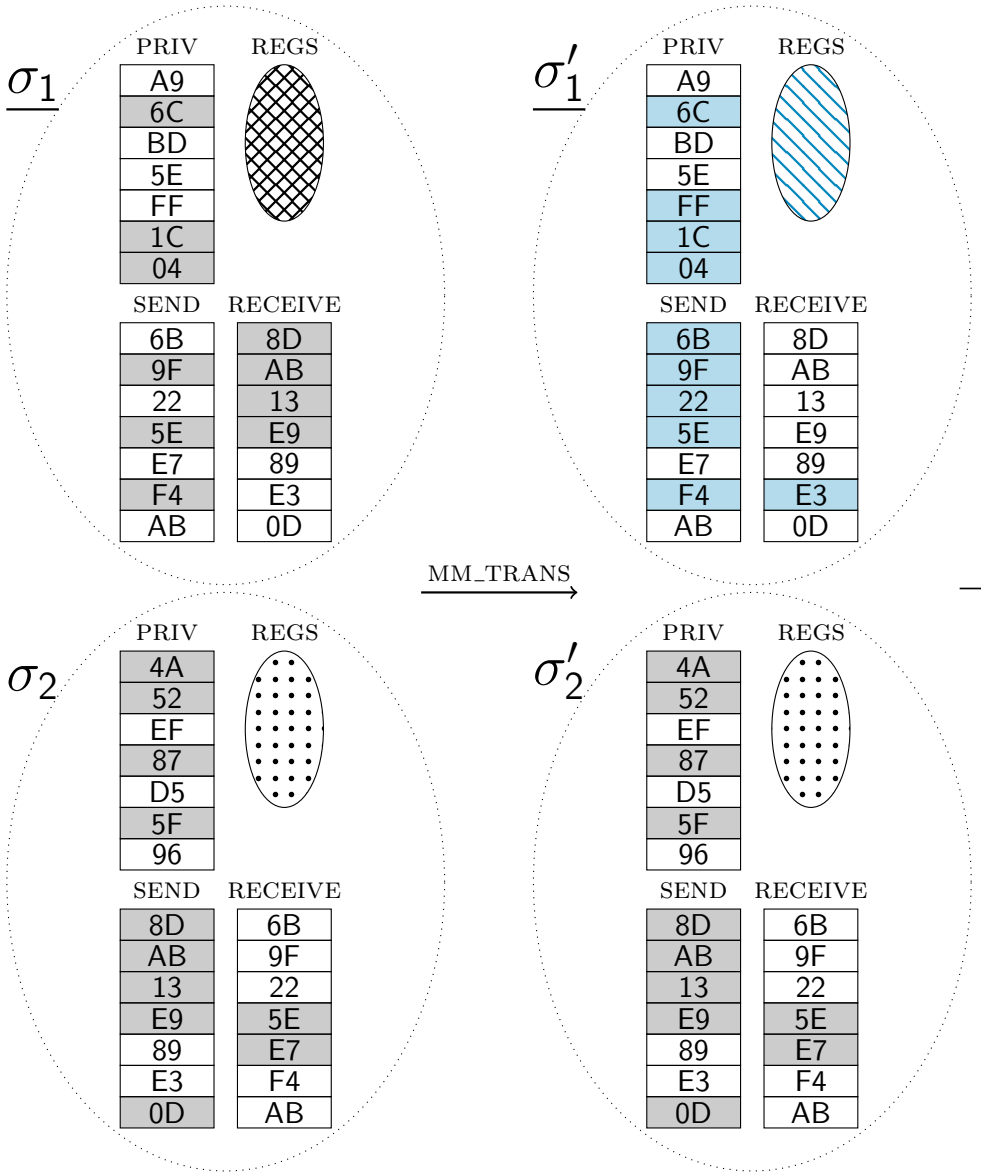
- $\sigma'_i.priv = \sigma_i.priv$
- $\forall k, \sigma'_i.s_k = \sigma_i.s_k$
- $\forall k \neq j, \sigma'_i.r_k = \sigma_i.r_k$

Abstract Guest Transition



$$\frac{\sigma'_i = run(\sigma_i) \qquad \forall k \neq i, \sigma'_k = \sigma_k[r_i \xleftarrow{VAL} \sigma'_i.s_k]}{\langle i, \boldsymbol{\sigma_1}, ...\boldsymbol{\sigma_n}\rangle \;\rightarrow\; \langle i, \boldsymbol{\sigma'_1}, ...\boldsymbol{\sigma'_n}\rangle}$$

Abstract Memory Management Transition



$$\frac{decode(\sigma_i.abs\_regs) = mm(\sigma'_i)}{\langle i, \sigma_1, ...\boldsymbol{\sigma_i}, ...\sigma_n\rangle \;\rightarrow\; \langle i, \sigma_1, ...\boldsymbol{\sigma'_i}, ...\sigma_n\rangle}$$

Concrete State

Concrete State:

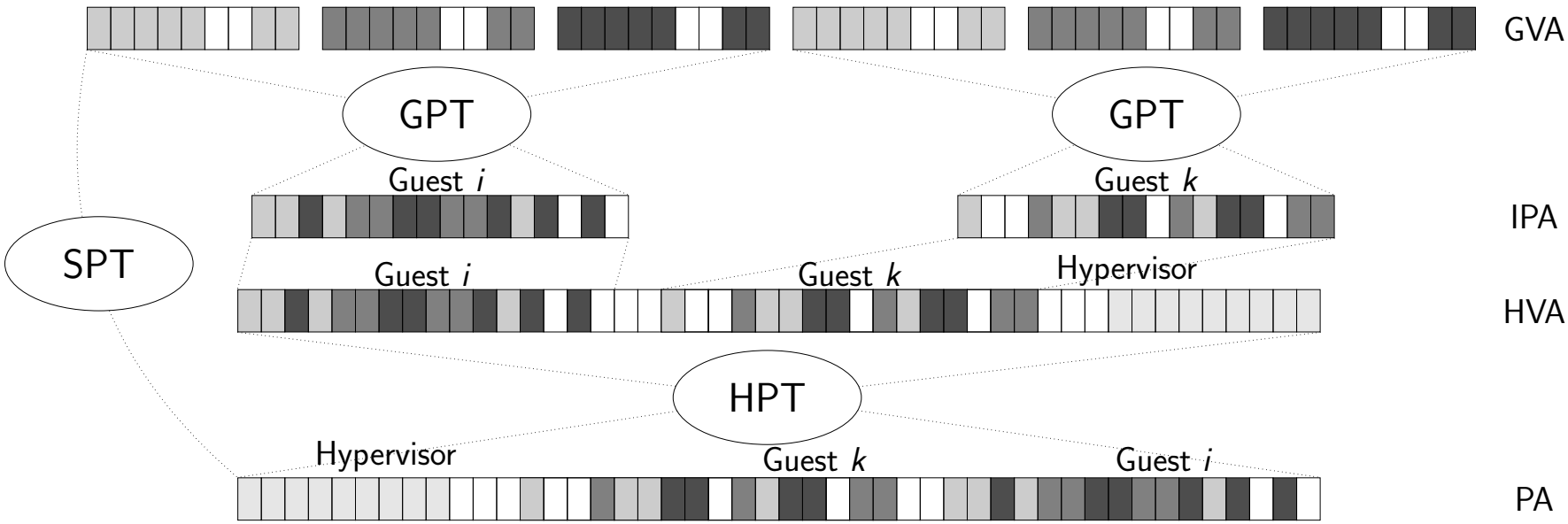
$$\sigma = \langle \sigma_{\mathrm{HW}}, \sigma_{\mathrm{HYP}}, \mathit{exception} \rangle$$

Where:

- $\sigma_{\mathrm{HW}} = \langle \mathit{mem}, \mathit{base}, \mathit{level}, \mathit{regs}_{gp}, \mathit{regs}_{mmu}, \mathit{regs}_{gic} \rangle$
- $\sigma_{\mathrm{HYP}} = \langle \mathit{curr}, \sigma_{\mathrm{int}}, \langle \sigma_{G1}, ..., \sigma_{Gn} \rangle \rangle$
- $\sigma_{Gi} = \langle \mathit{vbase}, \mathit{vmode}, \mathit{vbnk}, \mathit{vregs}_{\mathrm{gp}}, \mathit{vregs}_{\mathrm{mmu}}, \mathit{vregs}_{\mathrm{gic}} \rangle$

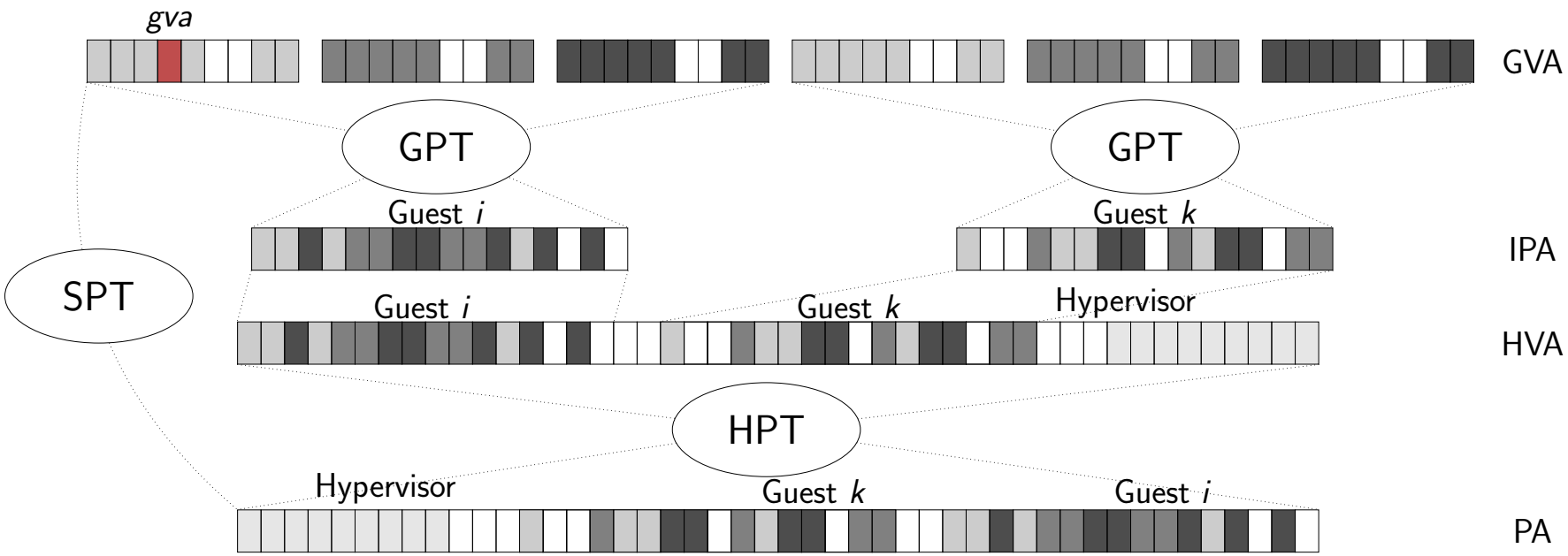
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



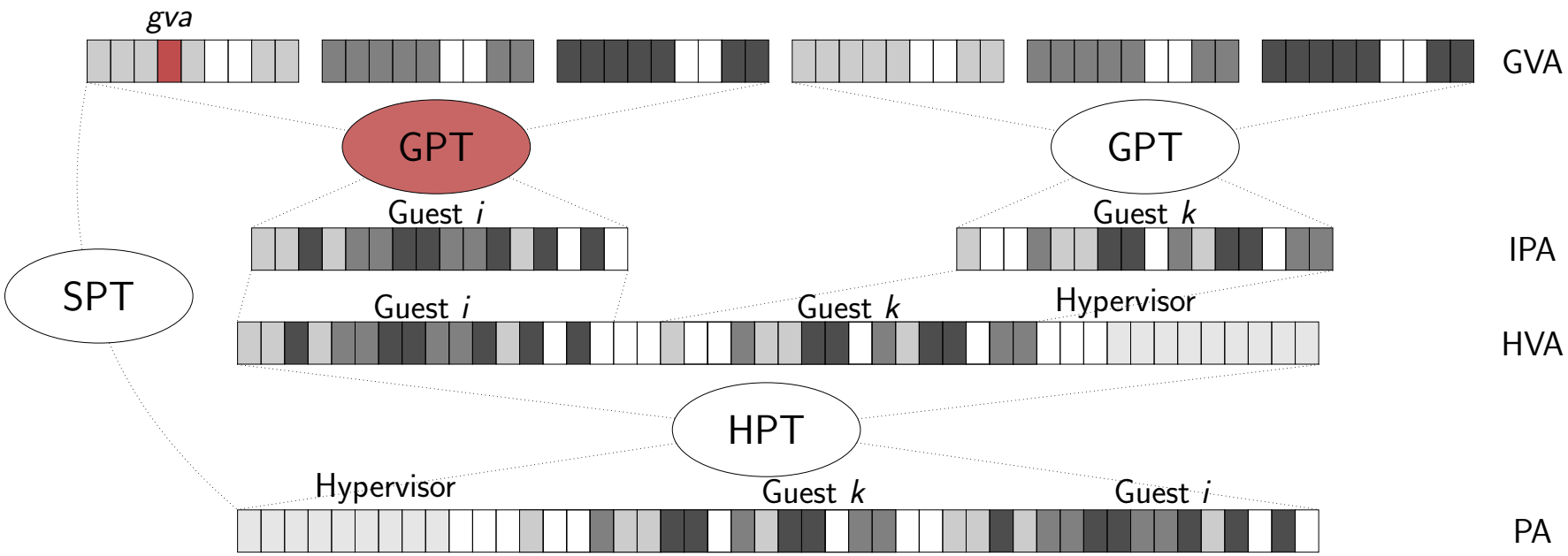
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



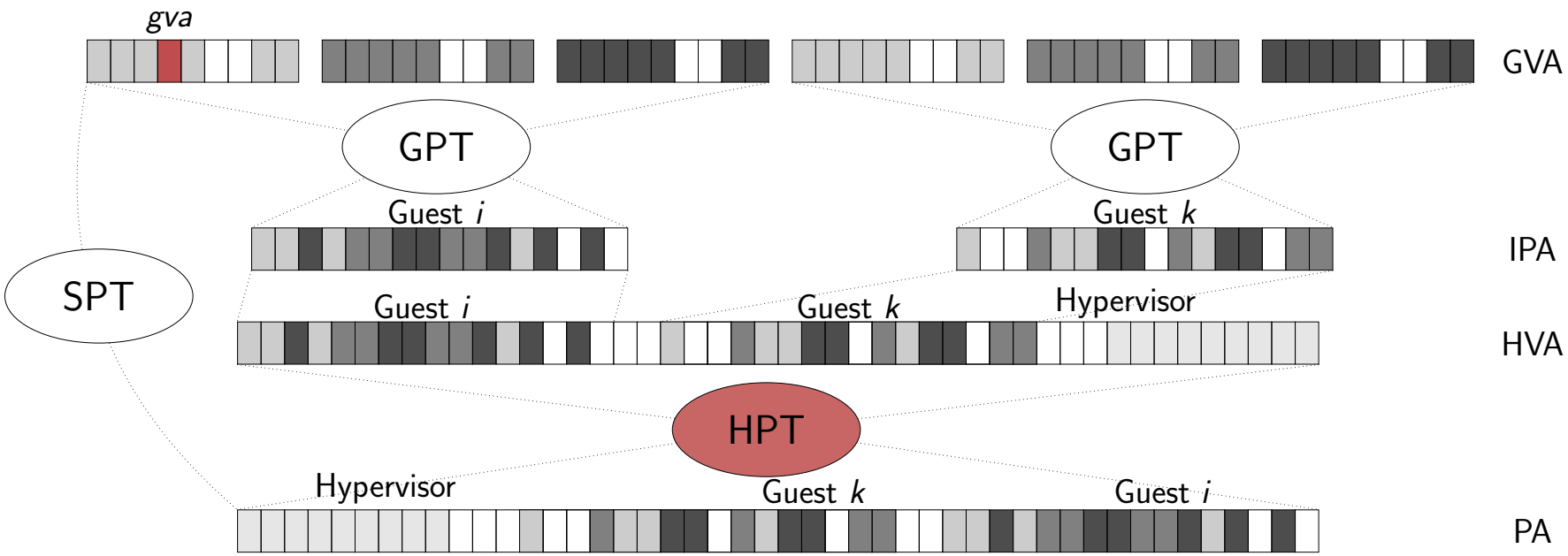
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



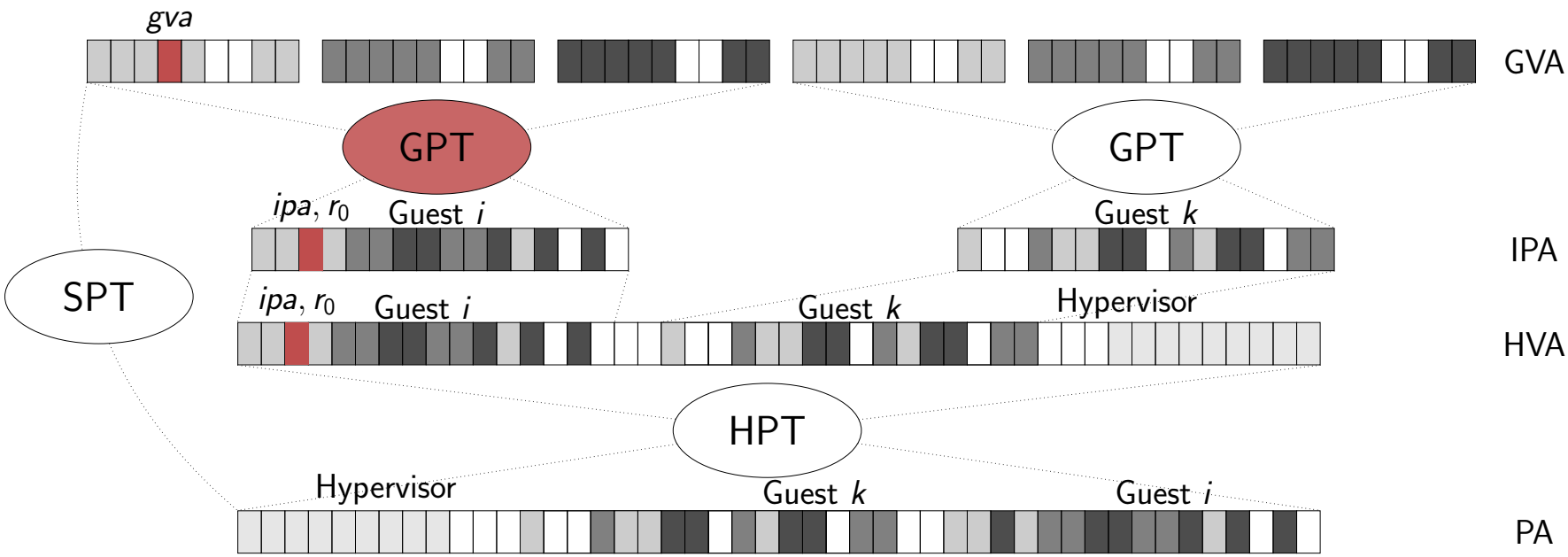
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



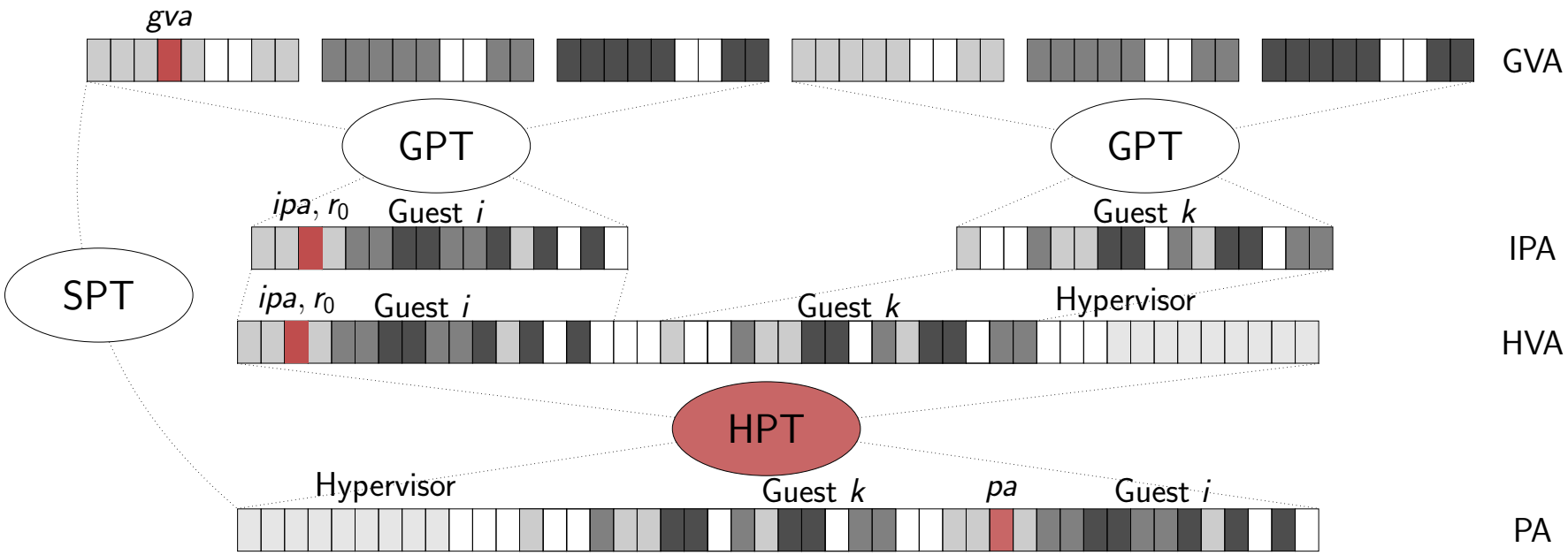
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



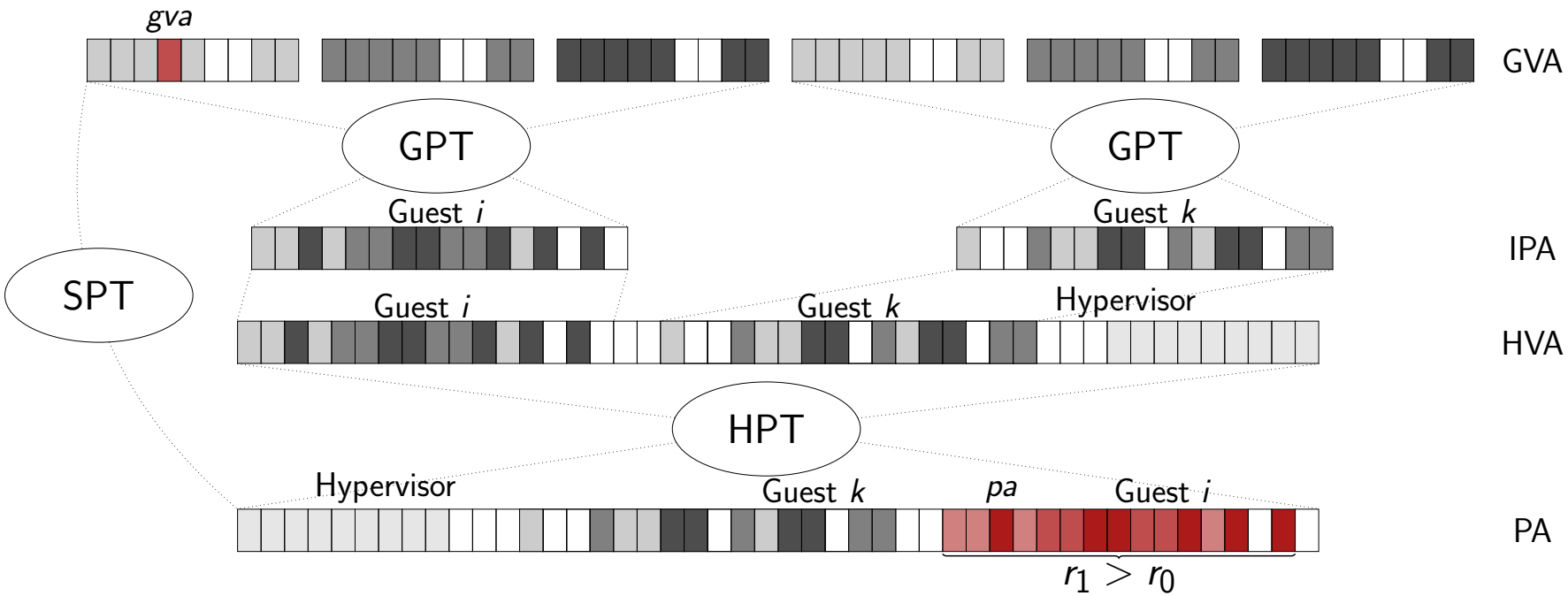
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ \mathbf{hpt(ipa) = (pa, \_)} \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathbf{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathbf{HYP}}} \\ abt \end{array} \right\rangle}$$



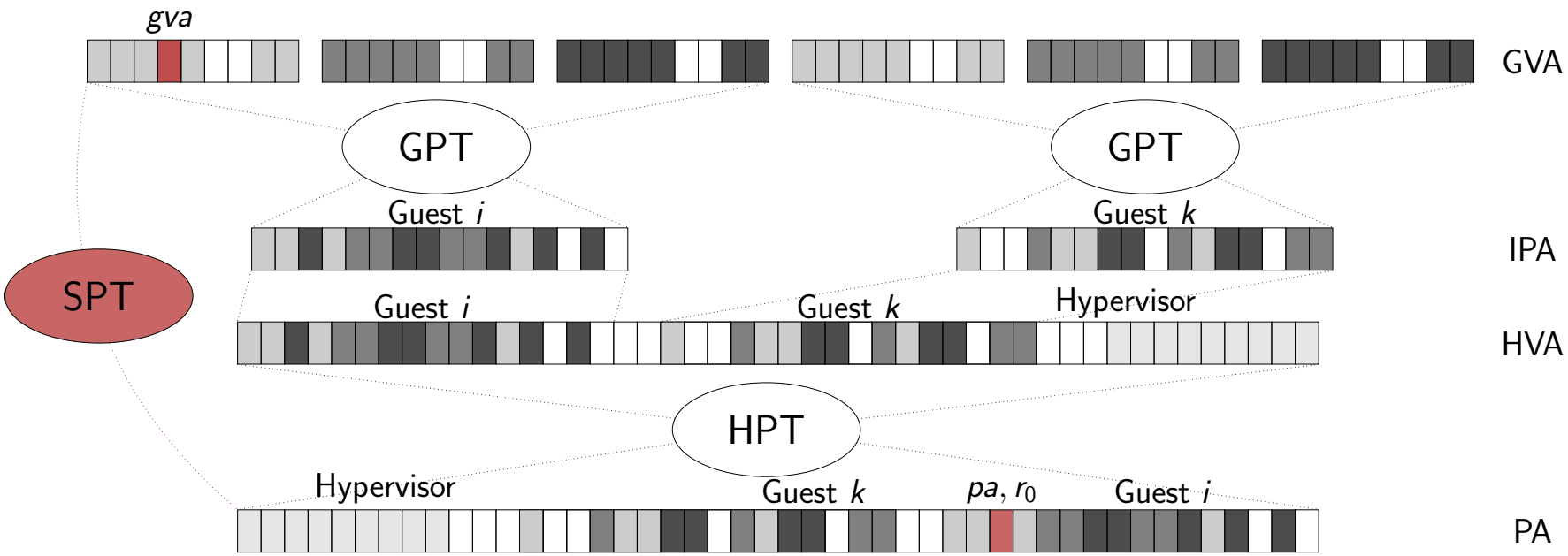
Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



Page Fault Transition:

$$\frac{\begin{array}{c} decode(abt, \sigma_{\mathrm{HW}}) = pf(gva) \qquad \sigma_{Gi}.vregs_{\mathrm{mmu}}.pg = enabled \\ hpt(\sigma_{Gi}.vbase) = (pbase, \_) \qquad \{(gva, (ipa, r_0))\} \in \Gamma_{pt(mem,pbase)} \\ hpt(ipa) = (pa, \_) \qquad \exists r_1 \geq r_0, allowed(pa, i, r_1) \\ \Gamma_{pt(mem',base)} = \Gamma_{pt(mem,base)} \cup \{(gva, (pa, r_0))\} \\ \sigma'_{\mathrm{HYP}} = \sigma_{\mathrm{HYP}}[\sigma_{\mathrm{int}} \leftarrow alloc(\sigma_{\mathrm{int}}, mem, gva)] \end{array}}{\left\langle \begin{array}{l} \langle \boldsymbol{mem}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle \rightarrow \left\langle \begin{array}{l} \langle \boldsymbol{mem'}, base, regs_{\mathrm{mmu}}, regs_{\mathrm{gp}}, pl1, regs_{\mathrm{gic}} \rangle \\ \boldsymbol{\sigma'_{\mathrm{HYP}}} \\ abt \end{array} \right\rangle}$$



Invariants on SPT

Property ensuring isolation:

if:

- $base \in \mathcal{B}_{\mathrm{SPT}}(\sigma_{\mathrm{int}}, i)$
- $(pa, r_0) \in Im(pt(mem, base))$

then:

- $\exists r_1 \geq r_0 \wedge allowed(pa, i, r_1)$

Invariants on SPT

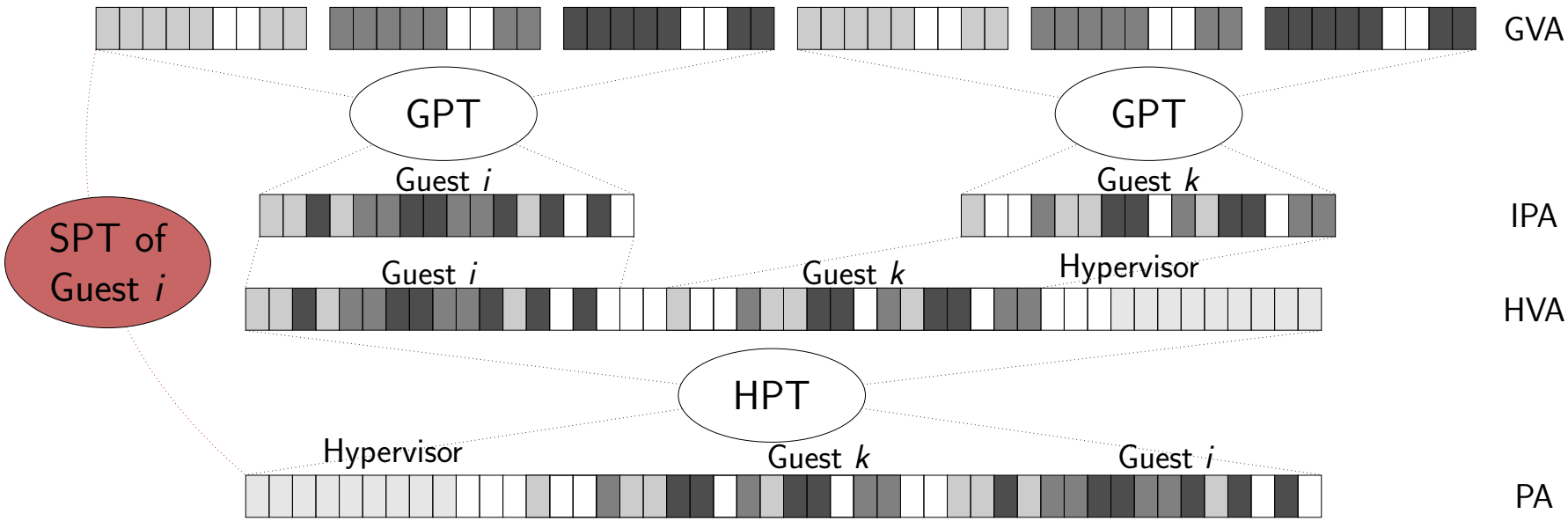
Property ensuring isolation:

if:

- $base \in \mathcal{B}_{\mathrm{SPT}}(\sigma_{\mathrm{int}}, i)$
- $(pa, r_0) \in Im(pt(mem, base))$

then:

- $\exists r_1 \geq r_0 \wedge allowed(pa, i, r_1)$



Invariants on SPT

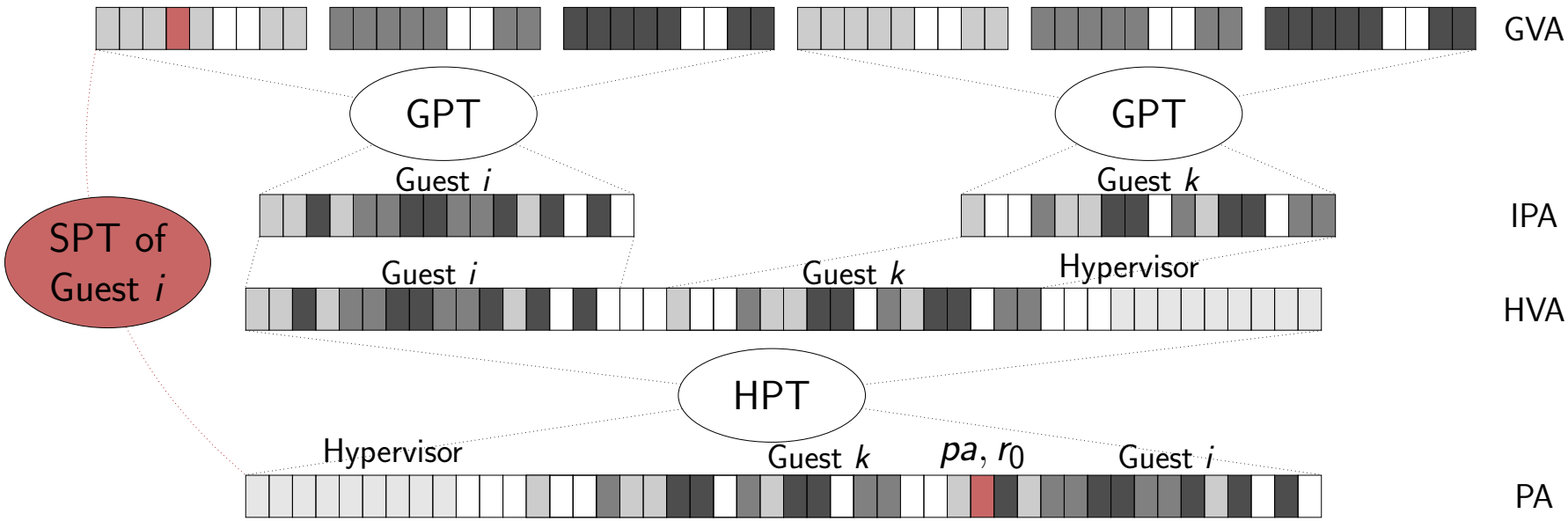
Property ensuring isolation:

if:

- $base \in \mathcal{B}_{\mathrm{SPT}}(\sigma_{\mathrm{int}}, i)$
- $(pa, r_0) \in Im(pt(mem, base))$

then:

- $\exists r_1 \geq r_0 \wedge allowed(pa, i, r_1)$



Invariants on SPT

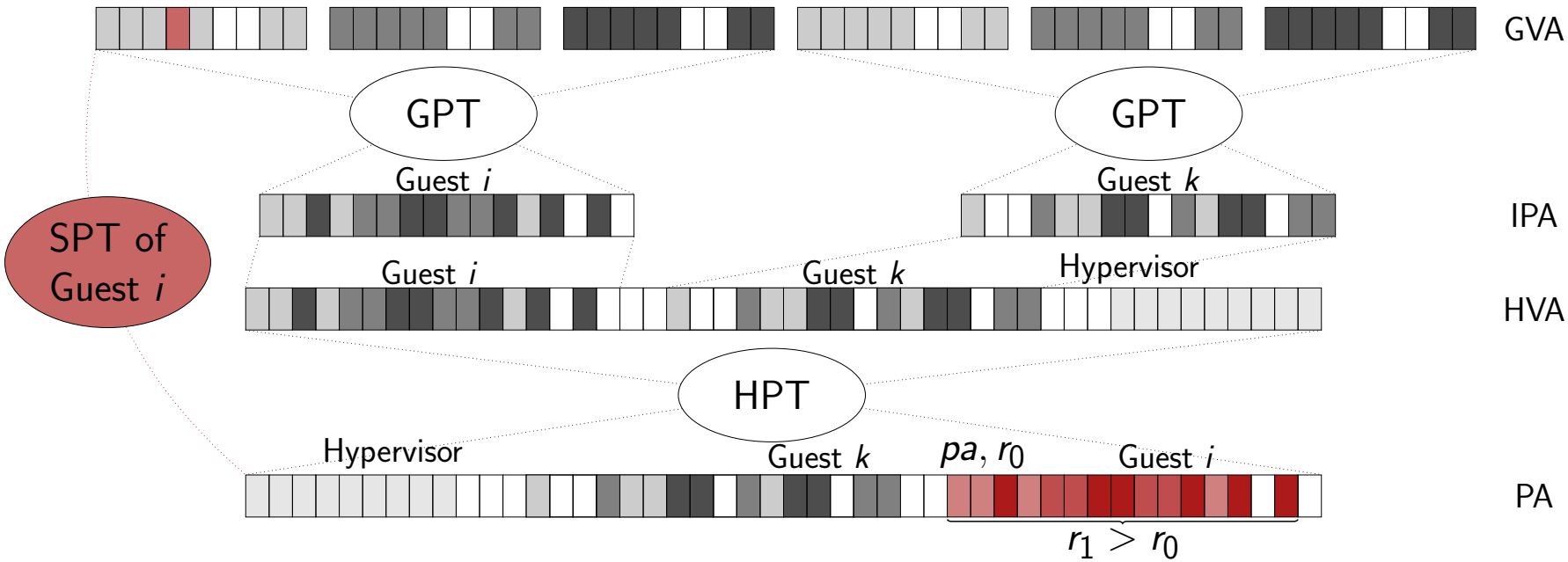
Property ensuring isolation:

if:

- $base \in \mathcal{B}_{\mathrm{SPT}}(\sigma_{\mathrm{int}}, i)$
- $(pa, r_0) \in Im(pt(mem, base))$

then:

- $\exists r_1 \geq r_0 \wedge allowed(pa, i, r_1)$



Outline

Characteristics of Prove & Run Tools

Smart:

- is a **functional** language.
- is **pure**.
- has an **imperative** syntax.
- allows to write both code and properties.

The Prover:

- is interactive.
- provides automatic resolution for simple goals.

Integrity Expressed in Smart

```
public lemma transition_integrity(state st, oracle o,
 idx i)
program
{{ state nst, seg priv, segstore send, segstore rec,
segment npriv, segstore nsend, segstore nrec, idx curr }}
{
  state@curr_guest(st, curr+);
  i != curr => // [i] is not the current guest
  valid(st) =>
  ? transition(st, o, nst+) =>

  ? glo_get_segments(st, i, priv+, send+, rec+) =>
  ? glo_get_segments(nst, i, npriv+, nsend+, nrec+) &&
  npriv = priv &&
  nsend = send &&
  differ_only_on_curr(rec, nrec, curr);
}
```

Integrity expressed in Smart

"If

- *curr* is the index of the running guest.
- i is a guest index such that $i \neq curr$.
- $\langle curr, \sigma_1, ..., \sigma_i, ..., \sigma_n \rangle \rightarrow \langle curr', \sigma'_1, ..., \sigma'_i, ..., \sigma'_n \rangle$"

... in Smart :

```
state@curr_guest(st, curr+);
i != curr => // [i] is not the current guest
valid(st) =>
? transition(st, o, nst+) =>
```

Integrity expressed in Smart

"then

- $\sigma'_i.priv = \sigma_i.priv$
- $\forall k, \sigma'_i.s_k = \sigma_i.s_k$
- $\forall k \neq curr, \sigma'_i.r_k = \sigma_i.r_k$ "

... in Smart :

```
? glo_get_segments(st, i, priv+, send+, rec+) =>
? glo_get_segments(nst, i, npriv+, nsend+, nrec+) &&
npriv = priv &&
nsend = send &&
differ_only_on_curr(rec, nrec, curr);
```

Implementation Details

Measure: a *hint* means an interaction with the prover.

State of the proof:

- Properties of the **concrete** level: 5756 hints.
  - 3413 of which for the proof of preservation of invariants over the map operation.
- Proof of isolation at the abstract level: 336 hints for **confidentiality** and 235 hints for **integrity**.
- Proof of commutation of the **Guest transition**:1779 hints.
- Proof of commutation of the **Page Fault transition**:1885 hints.

Conclusions

Recent advances of Formal Methods in complex systems:

- Sel4, PROSPER, Verisoft XT project.

SPT management is:

- Complex.
- Error prone.

The work presented:

- Provides a **formalization** of a hypervisor, in particular of its **SPT management**.
- Studies the **properties** needed on SPT to ensure **isolation**.
- Aims at **simplifying further proofs** by linking the hypervisor model to an abstract model without SPT.

Thank you for your attention