Compositional Verification of Security Properties for Embedded Execution Platforms

Christoph Baumann*, Oliver Schwarz*, Mads Dam*

*KTH Royal Institute of Technology, Stockholm, Sweden

*RISE.SICS, Kista, Sweden

cbaumann@kth.se

PROOFS, Taipei, 2017-09-29

Low and High Level System Security Bugs

'Dirty Cow' Linux vulnerability found after nine years

The 'Dirty Cow' bug was originally introduced nine years ago, and has been sitting unnoticed for much of that time

AFTER JEEP HACK, CHRYSLER RECALLS 1.4M VEHICLES FOR BUG FIX

Apple's App Store hit by the XCodeGhost of malware present

22 SEP 2015 4

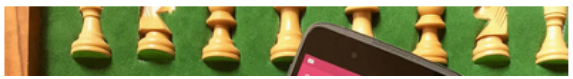
Apple, iOS, Malware, OS X, Vulnerability

'Quadrooter' flaws affect Android phones

All versions of Android are vulnerable to these fla until the September security release next month.

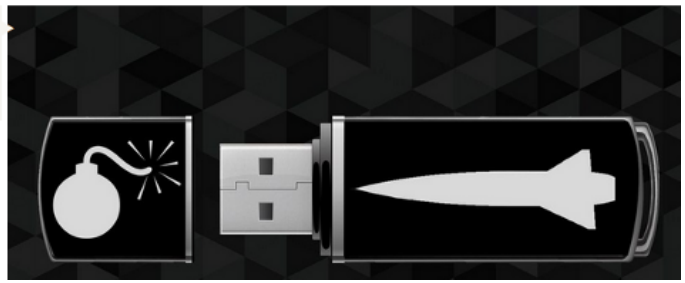
By Zack Whittaker for Zero Day | August 7, 2016 -- 22:00 GMT (23:00 BST) | Topic: Security

17 f 686 in 445



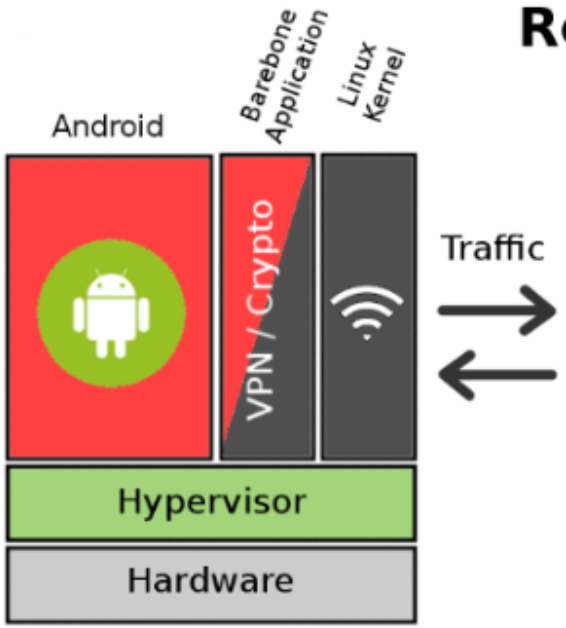
BadUSB Can Turn Thumb Drives Into Cyberweapons

2.9k SHARES 962 996 144 754 0 1



Tutus demonstrator

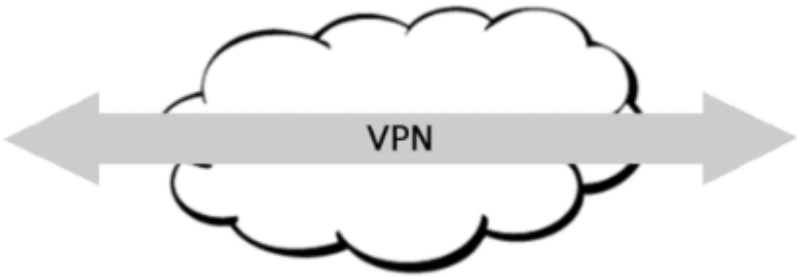
Red-black architecture



- Untrusted: Android and Linux Kernel
- Middle guest is trusted:
  - negotiates parameters for VPN
  - encrypts all outgoing traffic
  - decrypts all incoming traffic

System objectives:
- VPN is enforced under all conditions
- VPN parameters never compromised

FÄRIST MOBILE



FÄRIST VPN

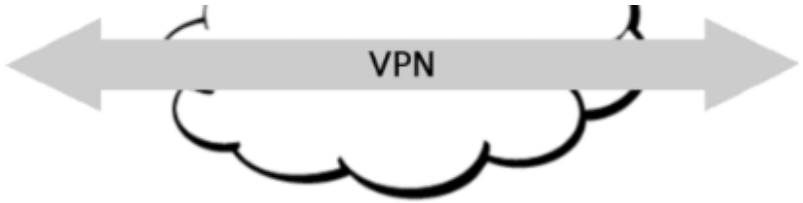
Tutus demonstrator

Red-black architecture

- Untrusted: Android and Linux Kernel

minimal COTS hypervisor for ARMv8:

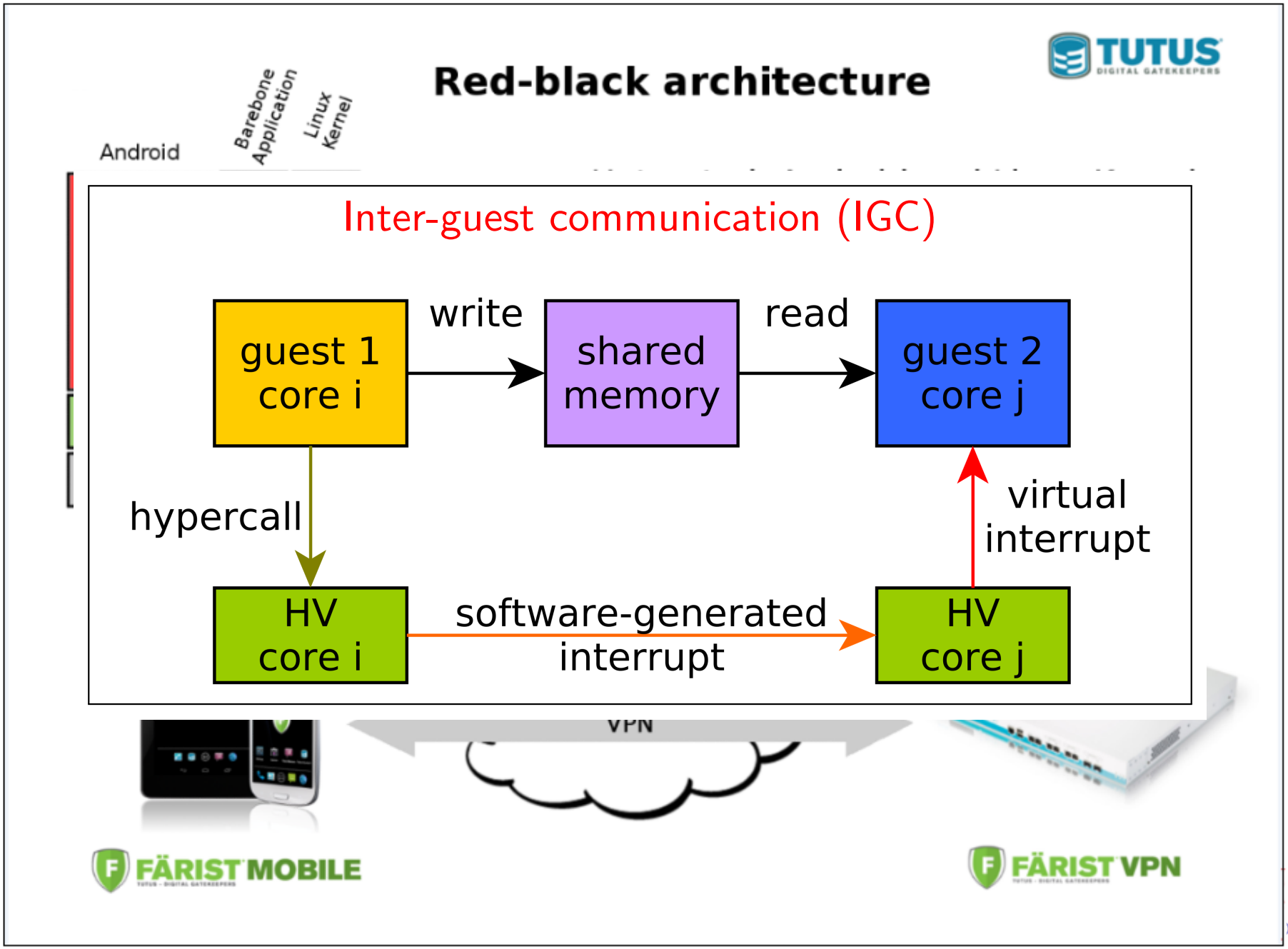
- fixed #guests, fixed memory size
- cores and devices owned exclusively
- no device virtualization, except: GIC
- secure boot loader
- memory isolation through HW extensions & SMMUs
- communication only through pre-defined channels



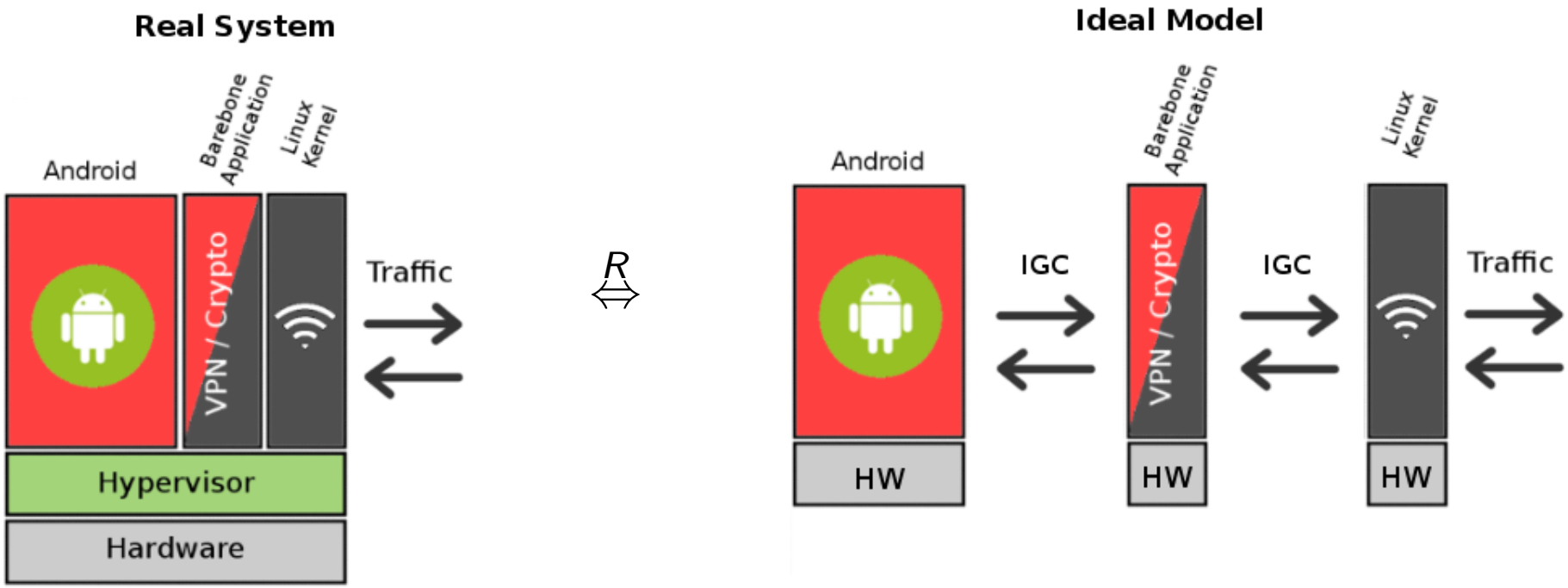
FÄRIST MOBILE

FÄRIST VPN

Tutus demonstrator

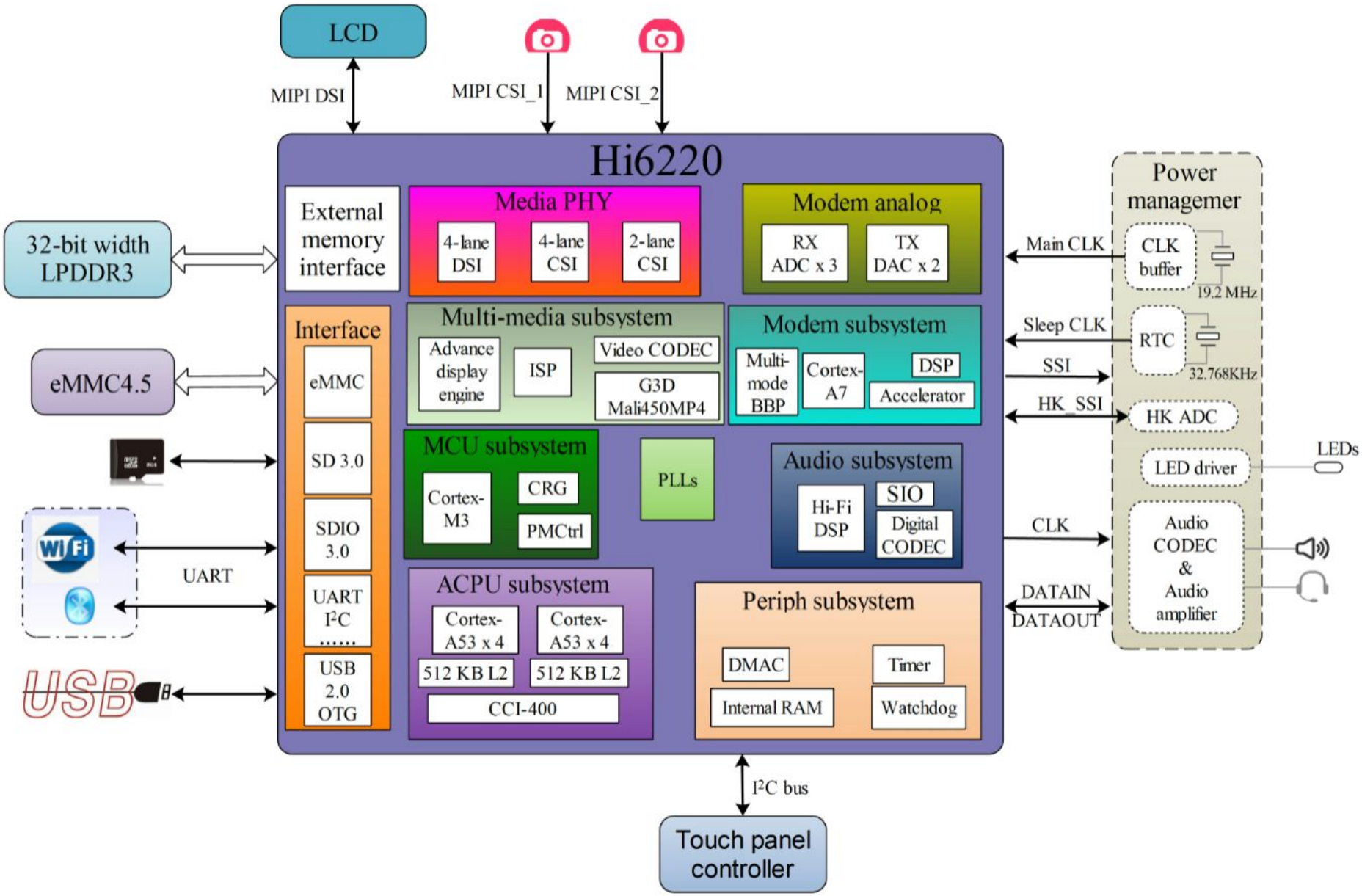


Goal: Bisimulation with Ideal Model

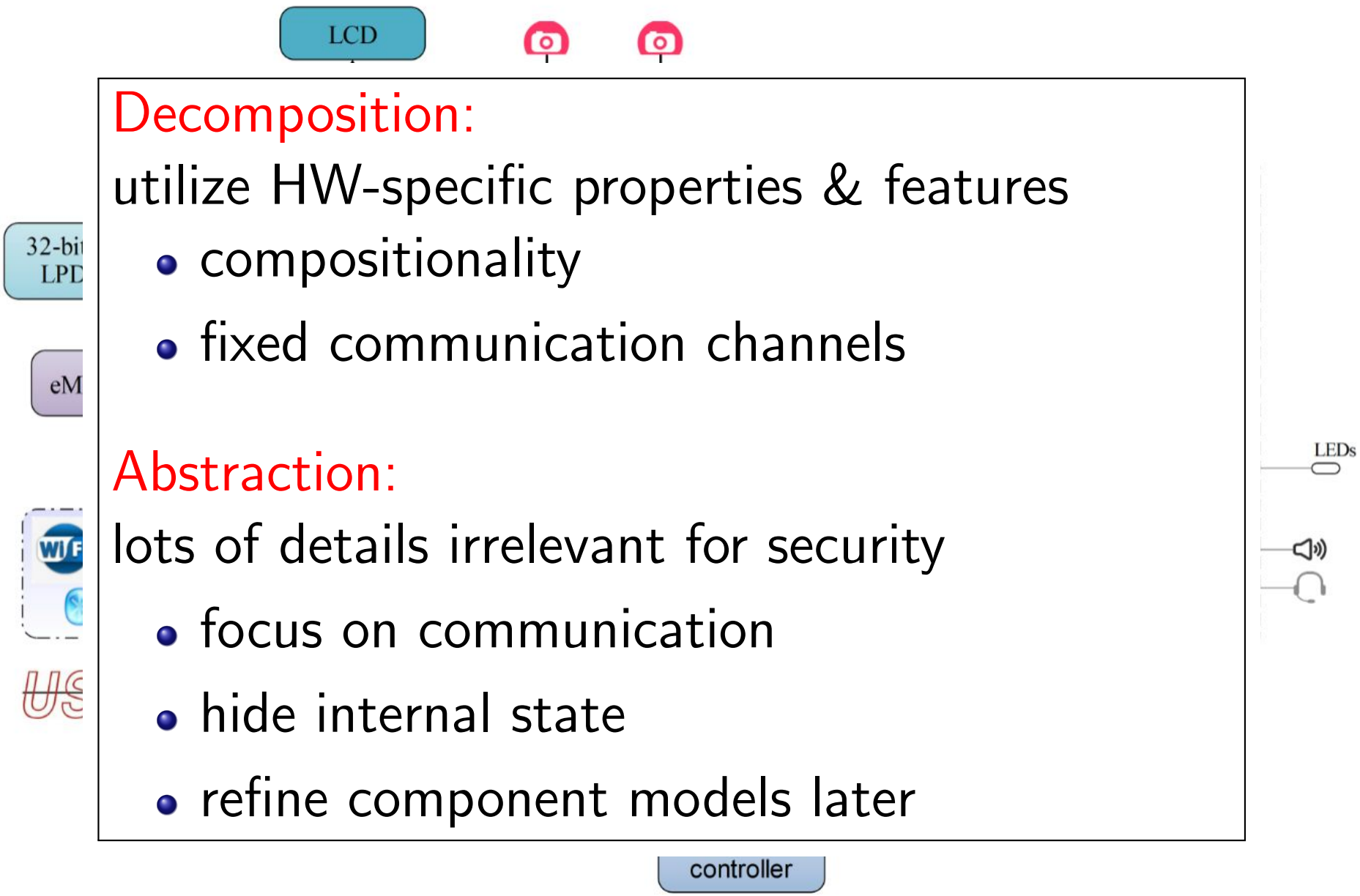


- ideal model: secure by construction
- bisimulation relation R: transfer information flow properties
- verification: focus on arbitrary guest steps here

SoCs complex / formal verification expensive



SoCs complex / formal verification expensive



Decomposition:

utilize HW-specific properties & features

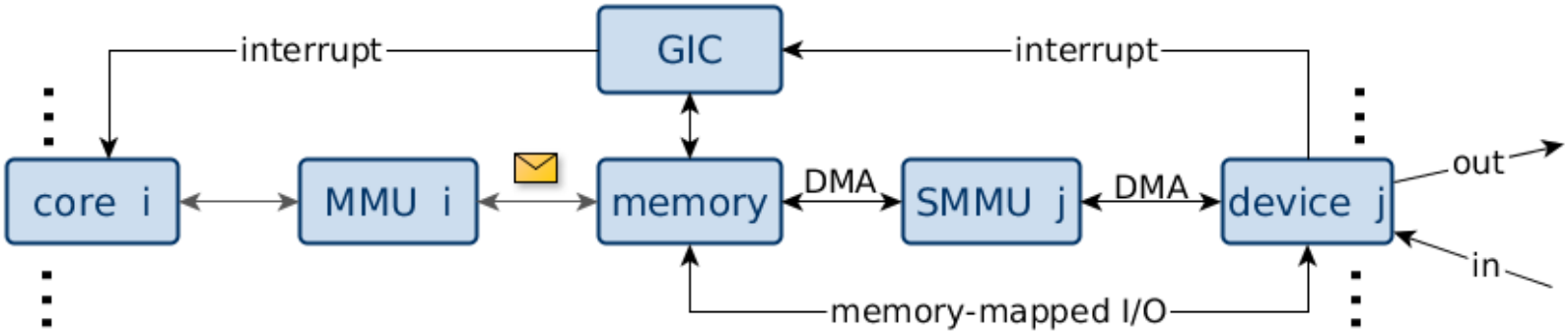
- compositionality
- fixed communication channels

Abstraction:

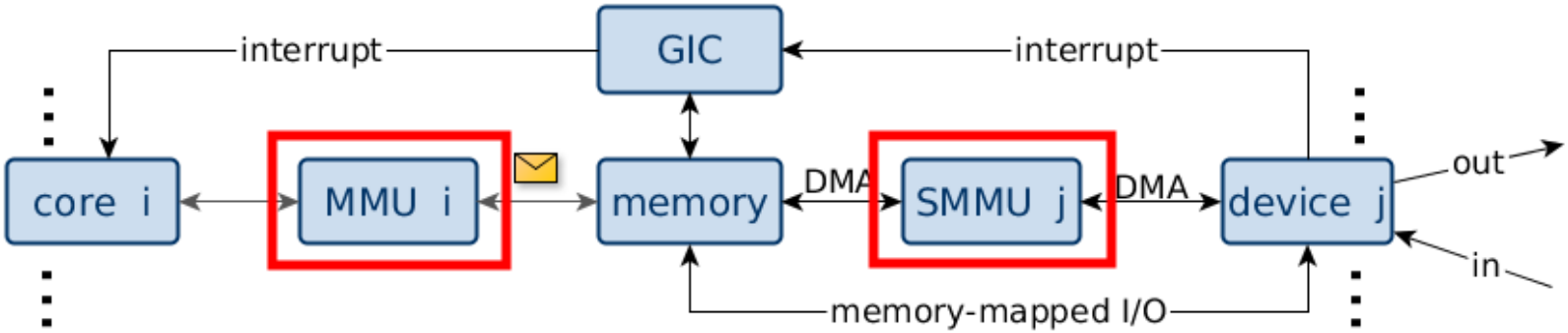
lots of details irrelevant for security

- focus on communication
- hide internal state
- refine component models later

ARMv8 platform model

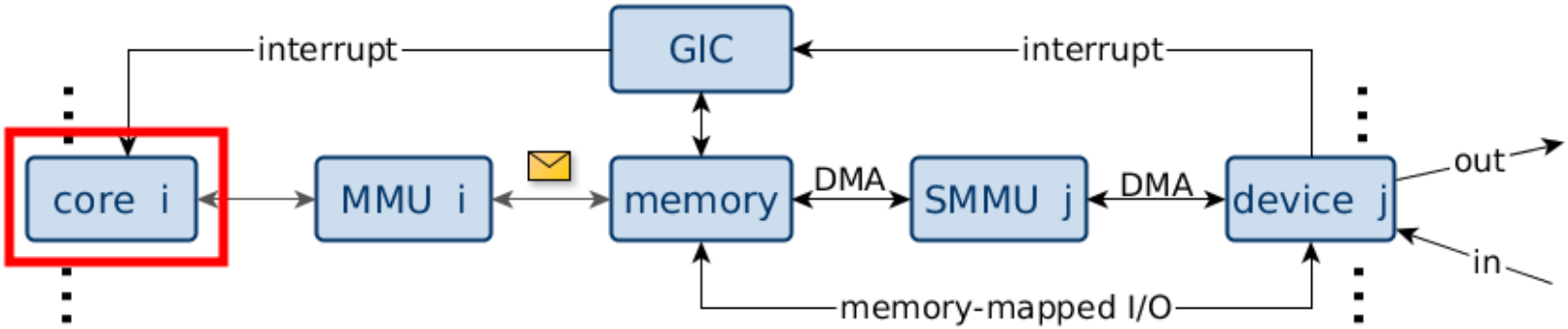


ARMv8 platform model



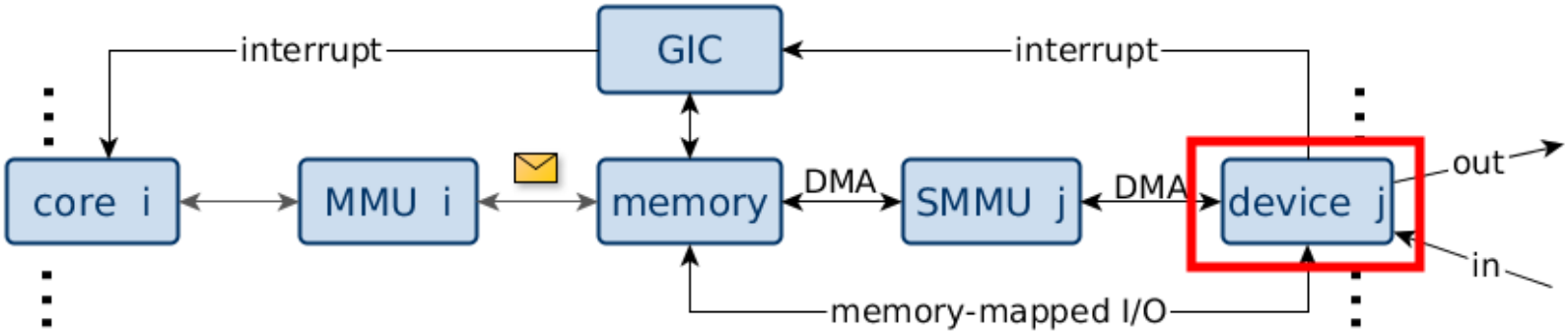
- (S)MMU: active?, page table base, current translations, mem requests

ARMv8 platform model



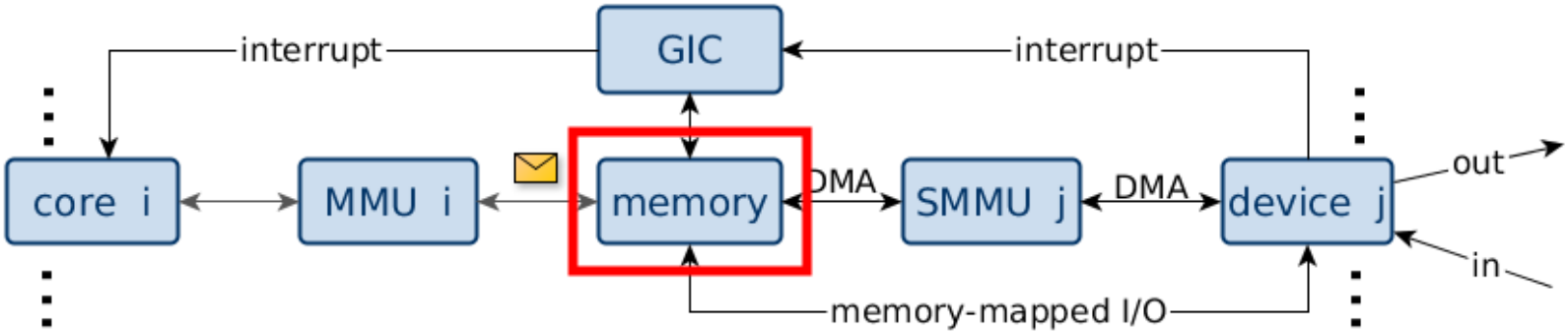
- (S)MMU: active?, page table base, current translations, mem requests
- Core: execution mode, some hypervisor registers relevant

ARMv8 platform model



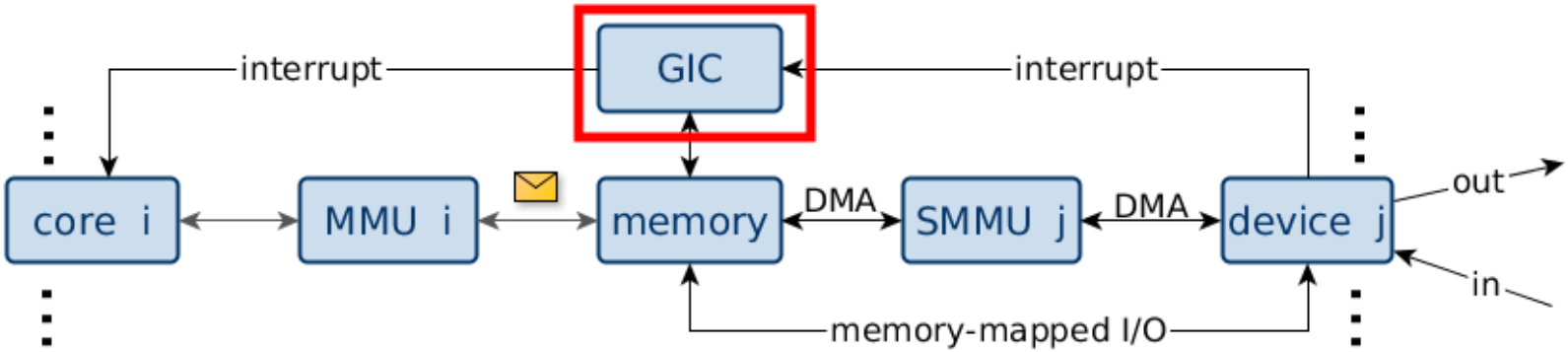
- (S)MMU: active?, page table base, current translations, mem requests
- Core: execution mode, some hypervisor registers relevant
- Device: mostly uninterpreted, DMA enabled?, track communication

ARMv8 platform model



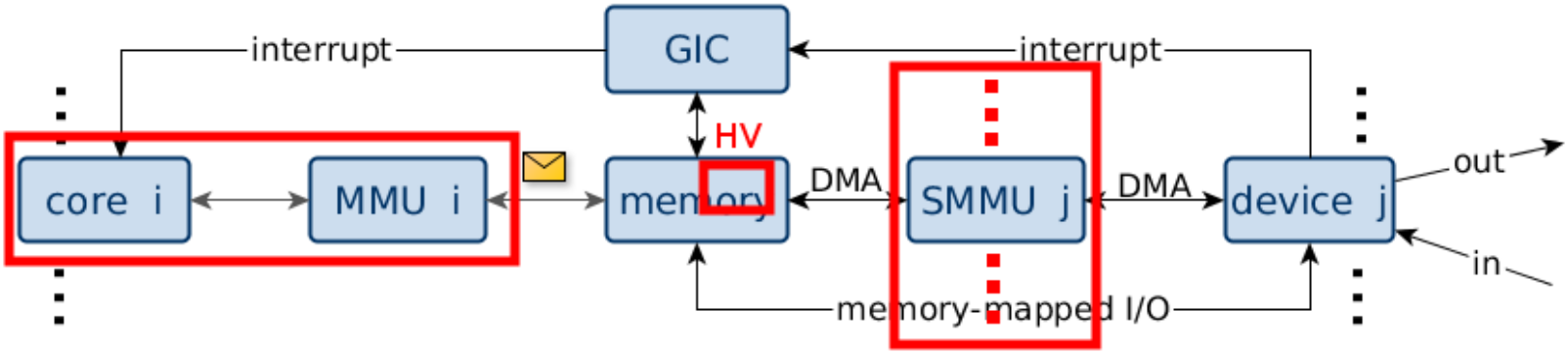
- (S)MMU: active?, page table base, current translations, mem requests
- Core: execution mode, some hypervisor registers relevant
- Device: mostly uninterpreted, DMA enabled?, track communication
- Memory: flat map of contents, received requests, forwarded I/O

ARMv8 platform model



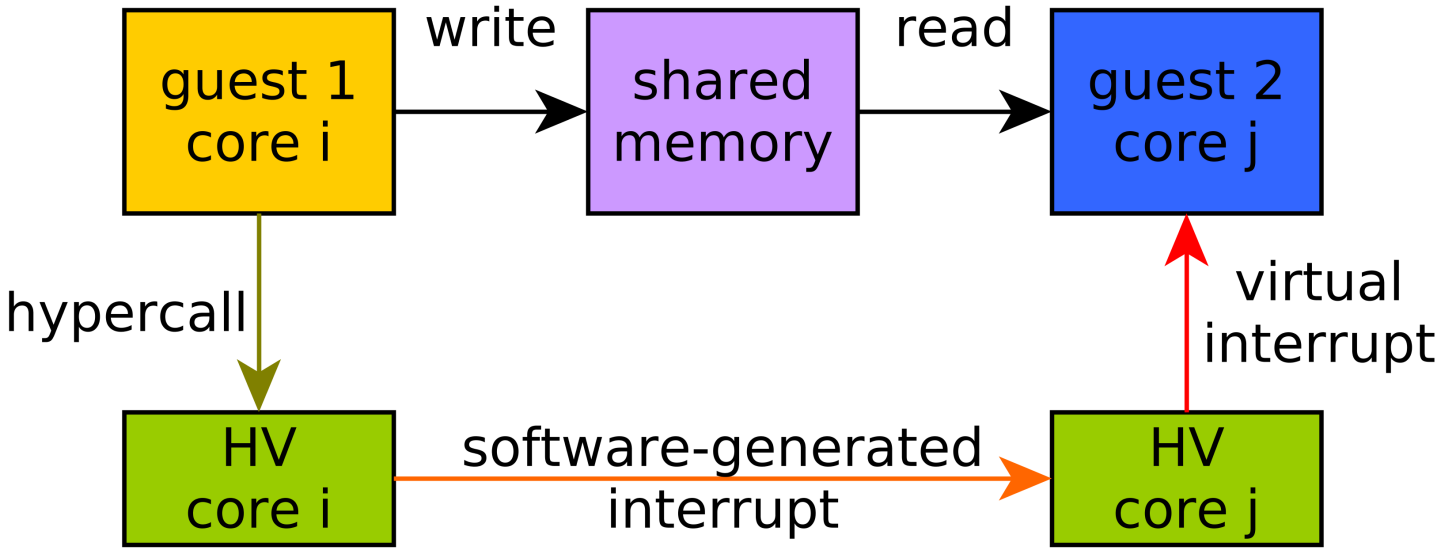
- (S)MMU: active?, page table base, current translations, mem requests
- Core: execution mode, some hypervisor registers relevant
- Device: mostly uninterpreted, DMA enabled?, track communication
- Memory: flat map of contents, received requests, forwarded I/O
- GIC: hypervisor-accessed registers, abstract interrupt state

ARMv8 platform model

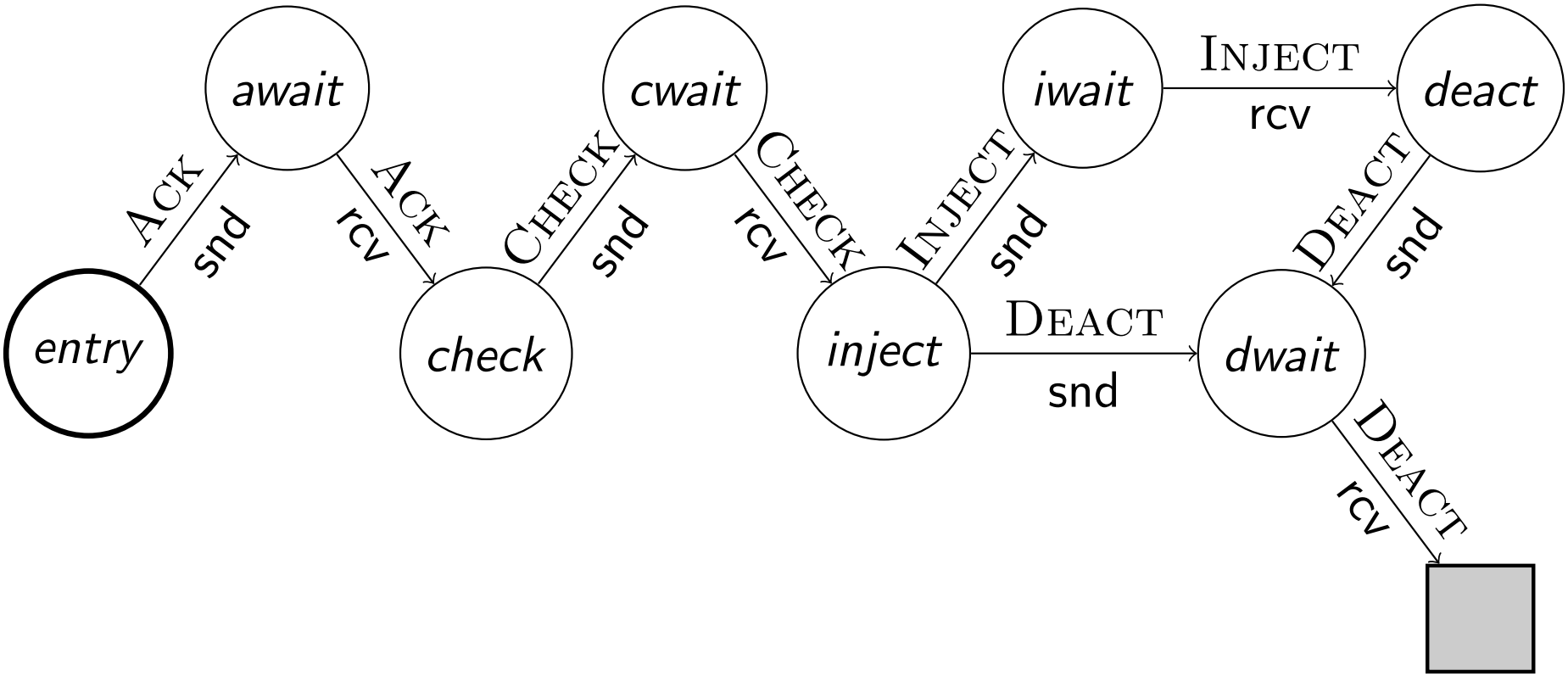


- (S)MMU: active?, page table base, current translations, mem requests
- Core: execution mode, some hypervisor registers relevant
- Device: mostly uninterpreted, DMA enabled?, track communication
- Memory: flat map of contents, received requests, forwarded I/O
- GIC: hypervisor-accessed registers, abstract interrupt state
- hypervisor: fine-grained LTS, communication with GIC

Hypervisor LTS: IGC interrupt injection



Hypervisor LTS: IGC interrupt injection



Verification: Platform Invariants

Component Constraints & HV configuration $\Rightarrow$ Invariant *Inv*:

- Messages & interrupts: preserve guest separation

Verification: Platform Invariants

Component Constraints & HV configuration $\Rightarrow$ Invariant *Inv*:

- Messages & interrupts: preserve guest separation
- Core: HV registers set up correctly, PC-safety in HV mode

Verification: Platform Invariants

Component Constraints & HV configuration $\Rightarrow$ Invariant *Inv*:

- Messages & interrupts: preserve guest separation
- Core: HV registers set up correctly, PC-safety in HV mode
- (S)MMU: active after init, points to right page table

Verification: Platform Invariants

Component Constraints & HV configuration $\Rightarrow$ Invariant *Inv*:

- Messages & interrupts: preserve guest separation
- Core: HV registers set up correctly, PC-safety in HV mode
- (S)MMU: active after init, points to right page table
- Device: inactive at boot

Verification: Platform Invariants

Component Constraints & HV configuration $\Rightarrow$ Invariant *Inv*:

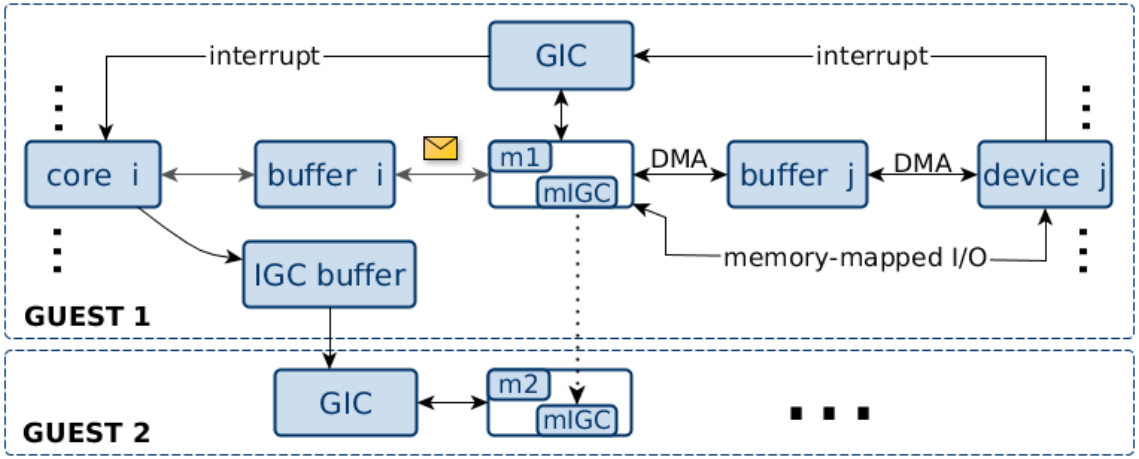
- Messages & interrupts: preserve guest separation
- Core: HV registers set up correctly, PC-safety in HV mode
- (S)MMU: active after init, points to right page table
- Device: inactive at boot
- Memory: correct page tables set up

Verification: Platform Invariants

Component Constraints & HV configuration $\Rightarrow$ Invariant *Inv*:

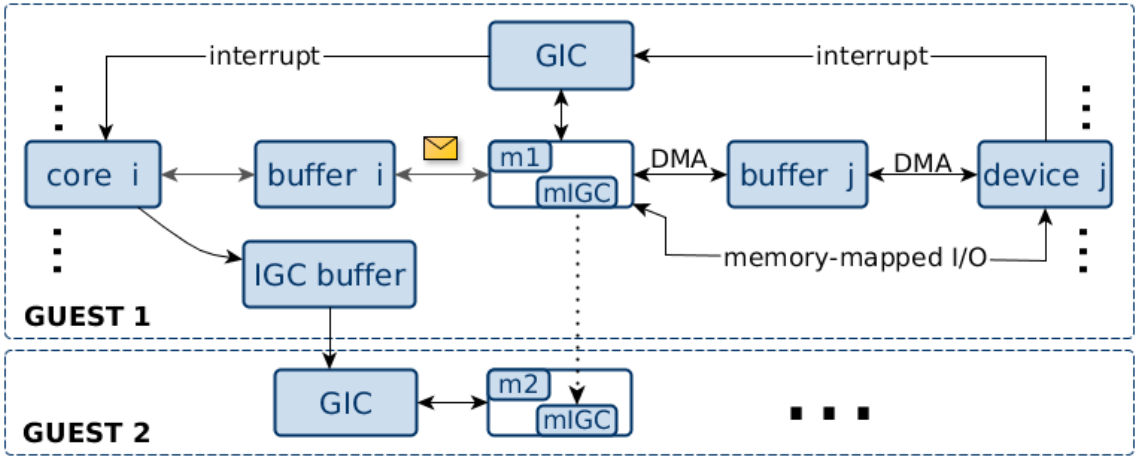
- Messages & interrupts: preserve guest separation
- Core: HV registers set up correctly, PC-safety in HV mode
- (S)MMU: active after init, points to right page table
- Device: inactive at boot
- Memory: correct page tables set up
- GIC: correct distributor configuration

Verification: Ideal Model



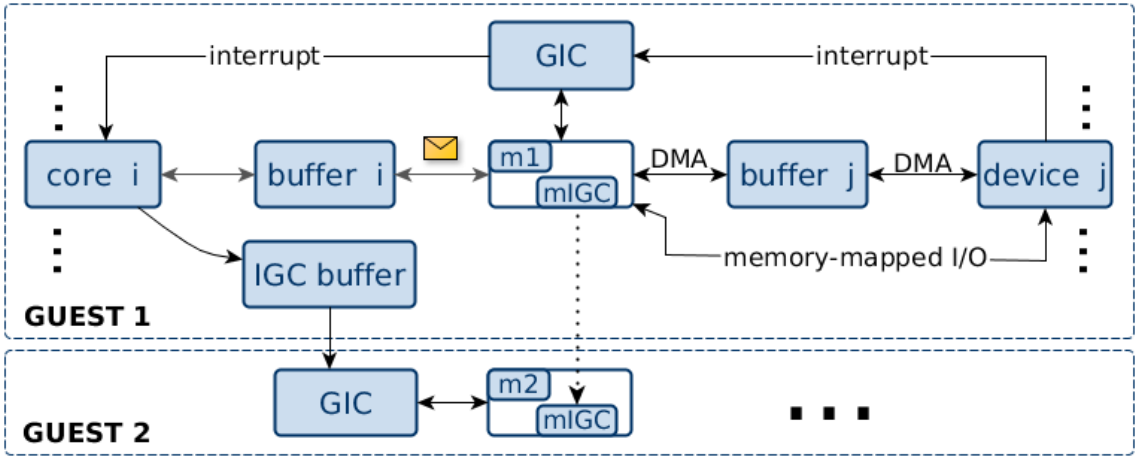
- ideal core: HV invisible / atomic hypercall semantics

Verification: Ideal Model



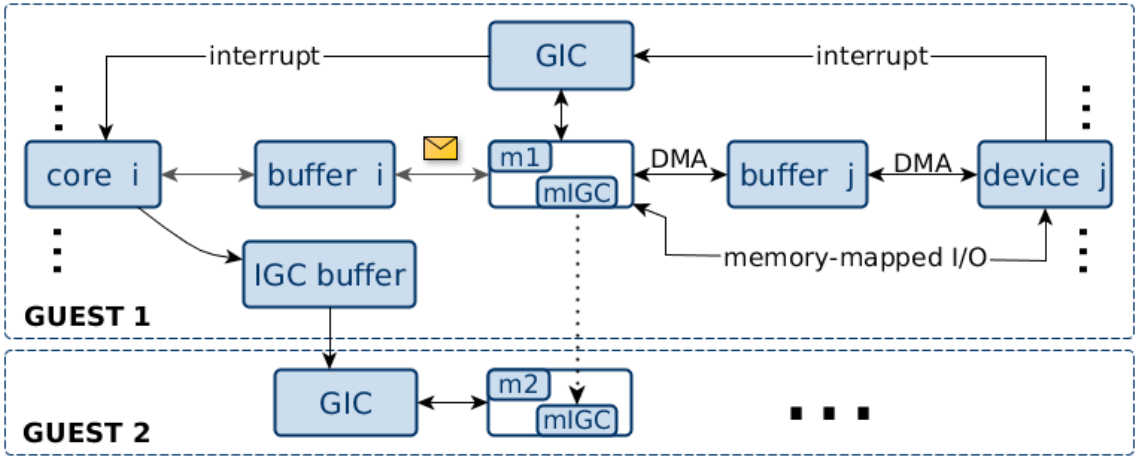
- ideal core: HV invisible / atomic hypercall semantics
- buffer for outgoing IGC notification interrupts

Verification: Ideal Model



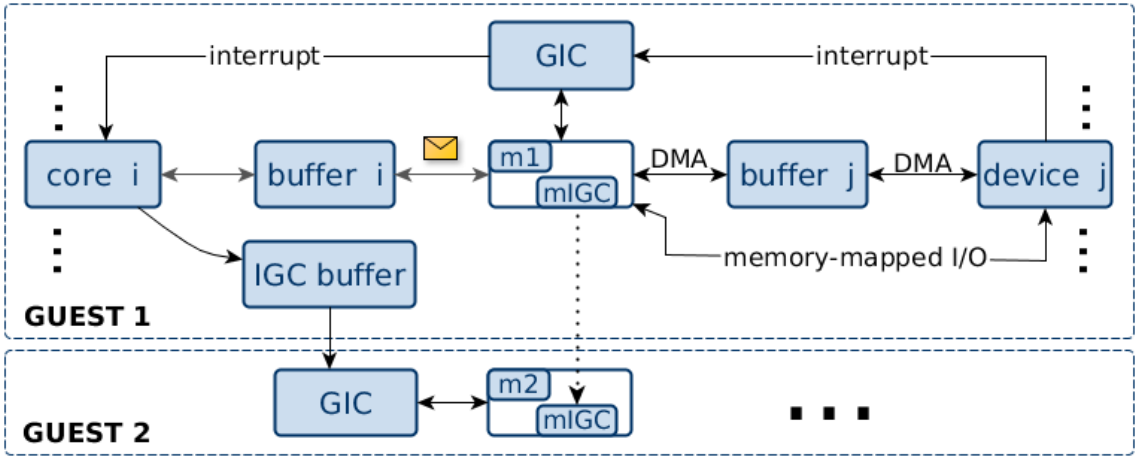
- ideal core: HV invisible / atomic hypercall semantics
- buffer for outgoing IGC notification interrupts
- IGC shared memory duplicated and copied on write

Verification: Ideal Model



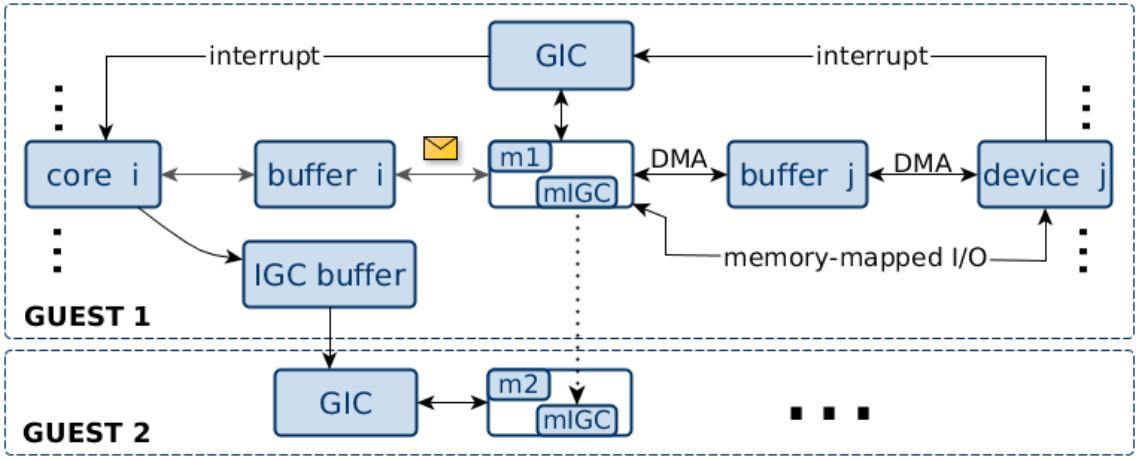
- ideal core: HV invisible / atomic hypercall semantics
- buffer for outgoing IGC notification interrupts
- IGC shared memory duplicated and copied on write
- ideal GIC: interrupt separation by construction

Verification: Ideal Model



- ideal core: HV invisible / atomic hypercall semantics
- buffer for outgoing IGC notification interrupts
- IGC shared memory duplicated and copied on write
- ideal GIC: interrupt separation by construction
- message buffers as placeholders for (S)MMUs

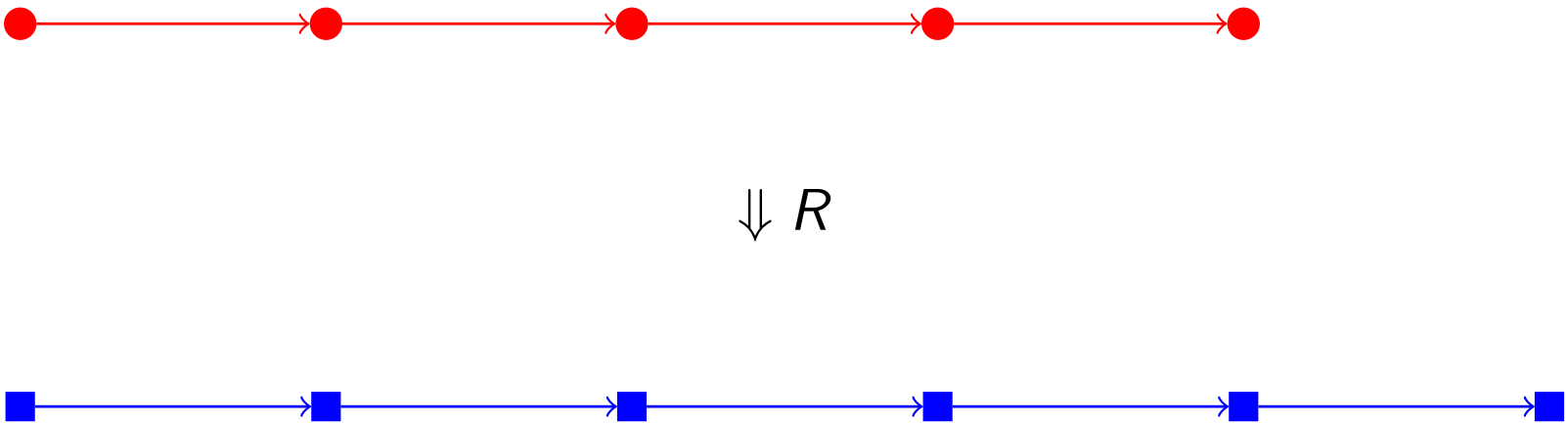
Verification: Ideal Model



- ideal core: HV invisible / atomic hypercall semantics
- buffer for outgoing IGC notification interrupts
- IGC shared memory duplicated and copied on write
- ideal GIC: interrupt separation by construction
- message buffers as placeholders for (S)MMUs
- memory: only guest portion, intermediate physical addresses

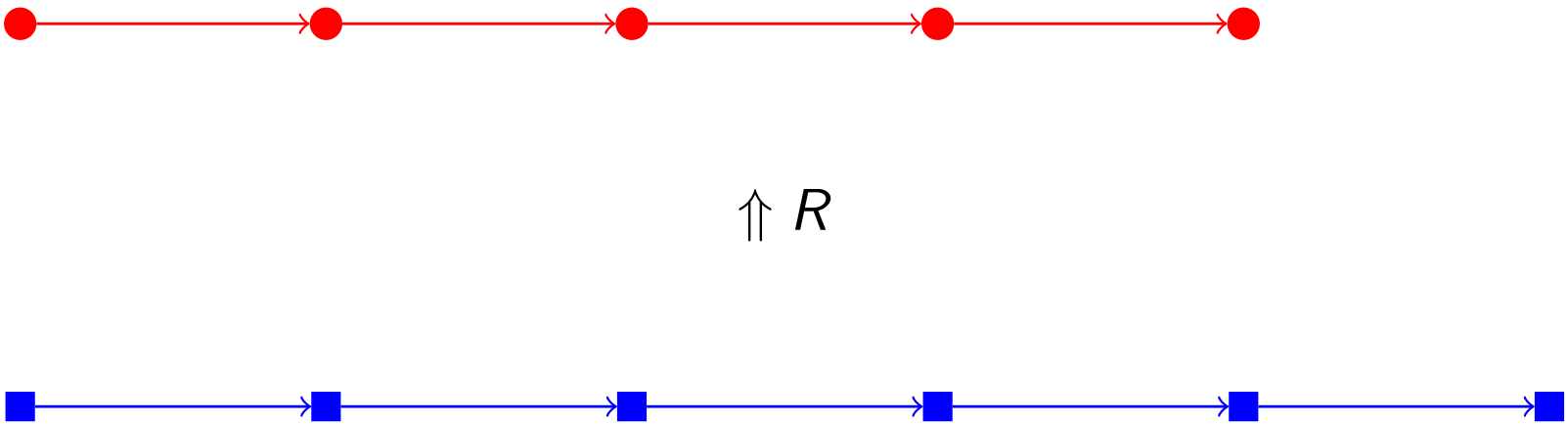
Verification: Bisimulation Theorem

Verification: Bisimulation Theorem

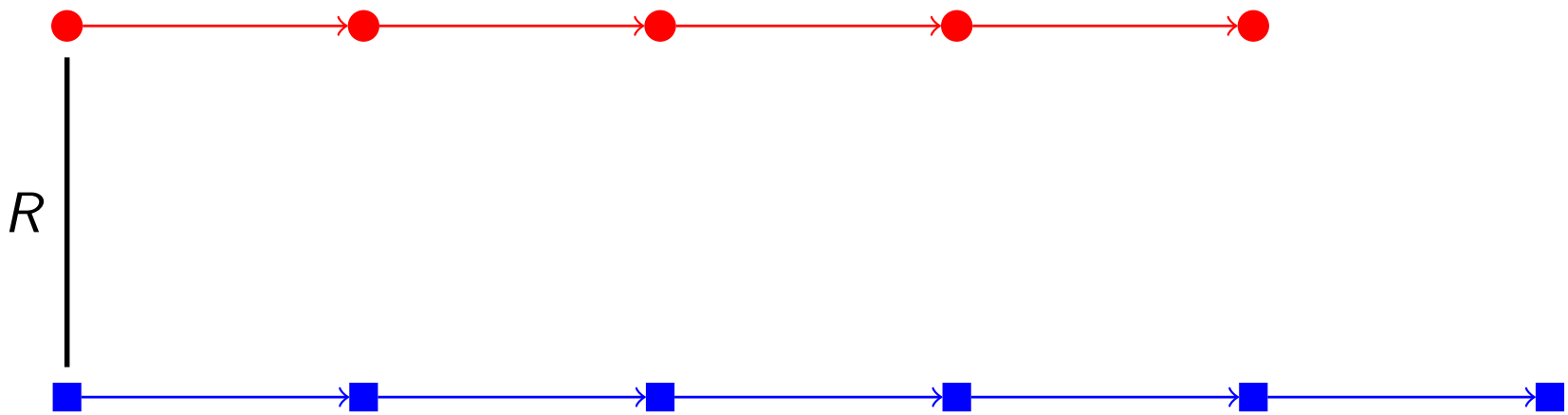


Verification: Bisimulation Theorem

Verification: Bisimulation Theorem



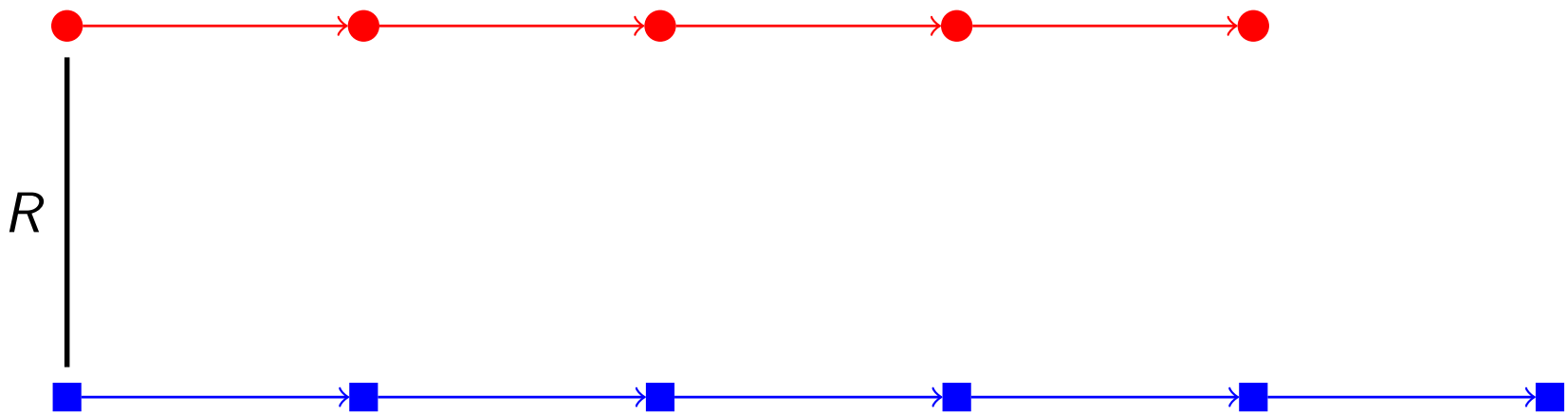
Verification: Bisimulation Theorem



Proof by induction on transition sequence:

- for any initial state σ_P^0, $Inv(\sigma_P^0)$ and exists σ_I^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for any initial state σ_I^0, exists σ_P^0 such that $\sigma_P^0\ R\ \sigma_I^0$

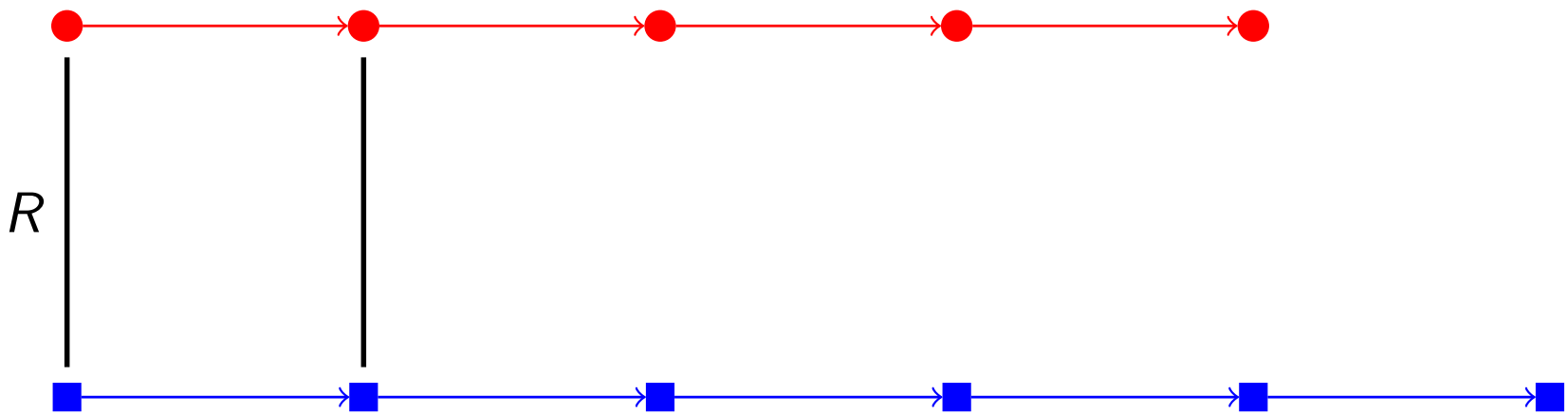
Verification: Bisimulation Theorem



Proof by induction on transition sequence:

- for any initial state σ_P^0, $Inv(\sigma_P^0)$ and exists σ_I^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for any initial state σ_I^0, exists σ_P^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for σ_P and σ_I with $\sigma_P\ R\ \sigma_I$ and $Inv(\sigma_P)$:
  - $\sigma_P \longrightarrow \sigma_P' \implies \exists \sigma_I'.\ \sigma_I \longrightarrow^* \sigma_I' \ \wedge\ \sigma_P'\ R\ \sigma_I'$
  - $\sigma_I \longrightarrow \sigma_I' \implies \exists \sigma_P'.\ \sigma_P \longrightarrow^* \sigma_P' \ \wedge\ \sigma_P'\ R\ \sigma_I'$

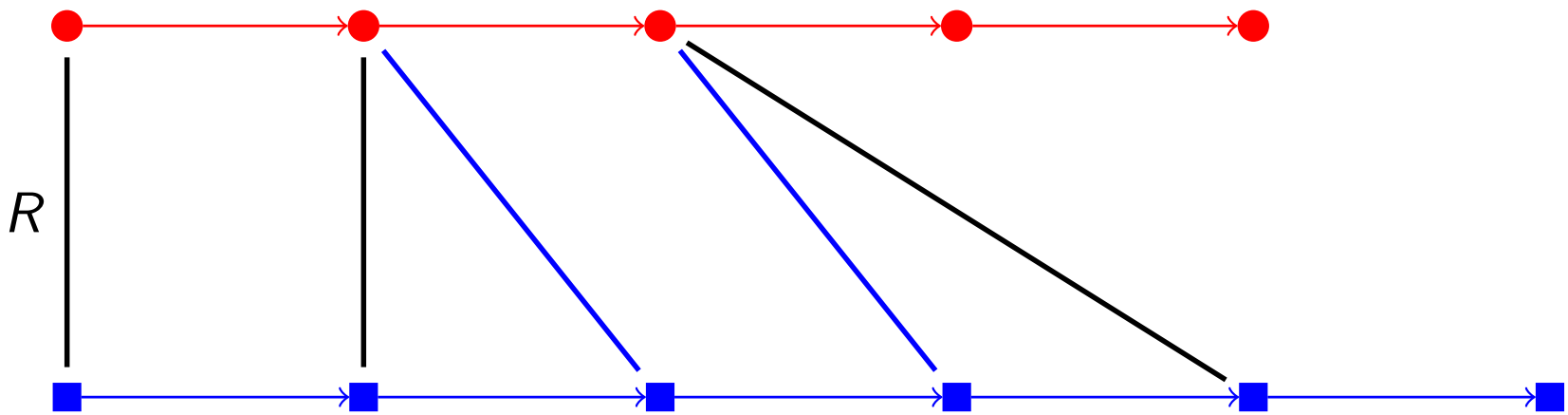
Verification: Bisimulation Theorem



Proof by induction on transition sequence:

- for any initial state σ_P^0, $Inv(\sigma_P^0)$ and exists σ_I^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for any initial state σ_I^0, exists σ_P^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for σ_P and σ_I with $\sigma_P\ R\ \sigma_I$ and $Inv(\sigma_P)$:
  - $\sigma_P \longrightarrow \sigma_P' \implies \exists \sigma_I'.\ \sigma_I \longrightarrow^* \sigma_I' \ \wedge\ \sigma_P'\ R\ \sigma_I'$
  - $\sigma_I \longrightarrow \sigma_I' \implies \exists \sigma_P'.\ \sigma_P \longrightarrow^* \sigma_P' \ \wedge\ \sigma_P'\ R\ \sigma_I'$

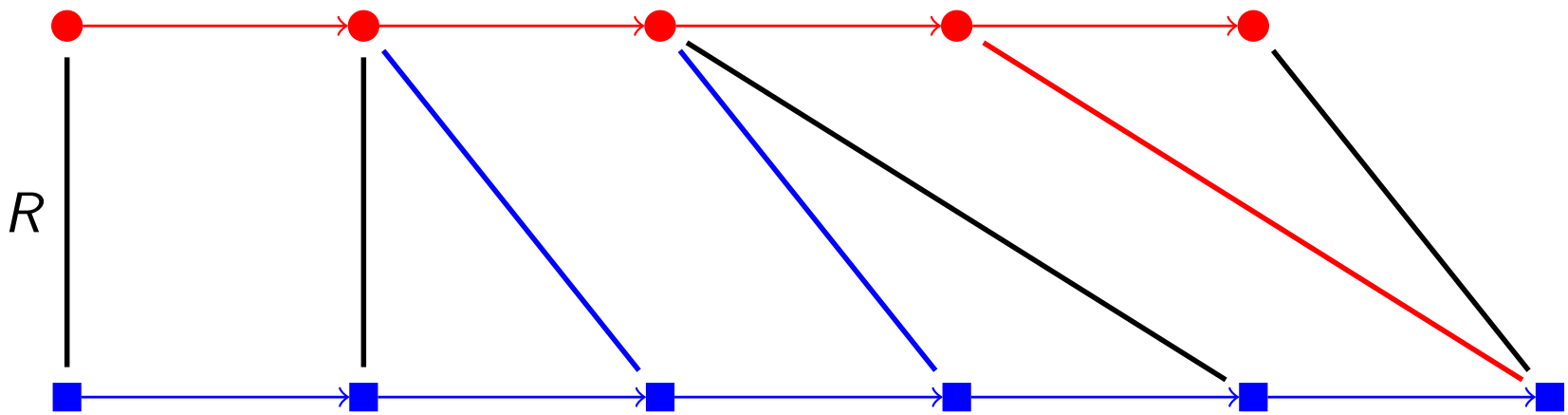
Verification: Bisimulation Theorem



Proof by induction on transition sequence:

- for any initial state σ_P^0, $Inv(\sigma_P^0)$ and exists σ_I^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for any initial state σ_I^0, exists σ_P^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for σ_P and σ_I with $\sigma_P\ R\ \sigma_I$ and $Inv(\sigma_P)$:
  - $\sigma_P \longrightarrow \sigma_P' \implies \exists \sigma_I'.\ \sigma_I \longrightarrow^* \sigma_I' \ \wedge\ \sigma_P'\ R\ \sigma_I'$
  - $\sigma_I \longrightarrow \sigma_I' \implies \exists \sigma_P'.\ \sigma_P \longrightarrow^* \sigma_P' \ \wedge\ \sigma_P'\ R\ \sigma_I'$

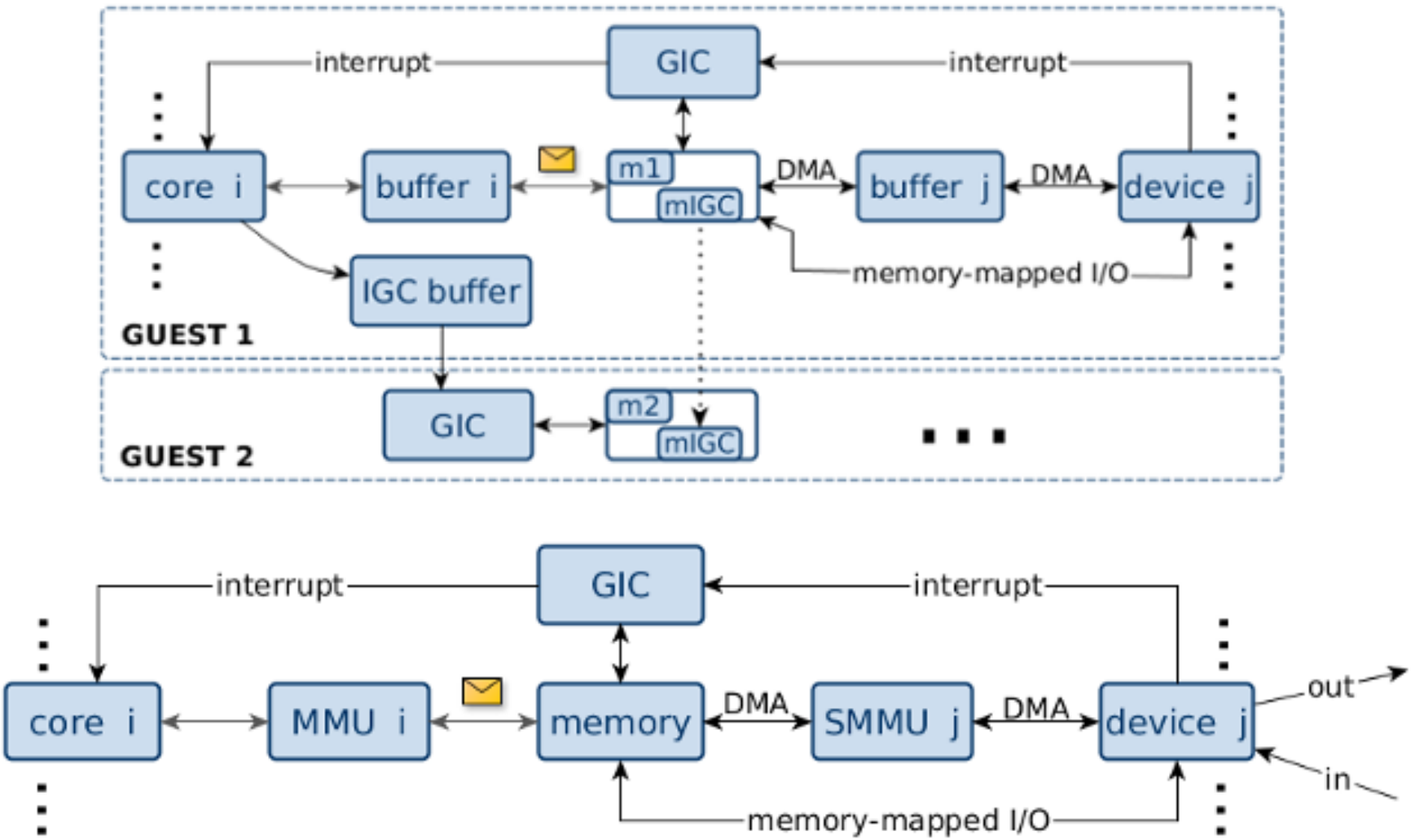
Verification: Bisimulation Theorem



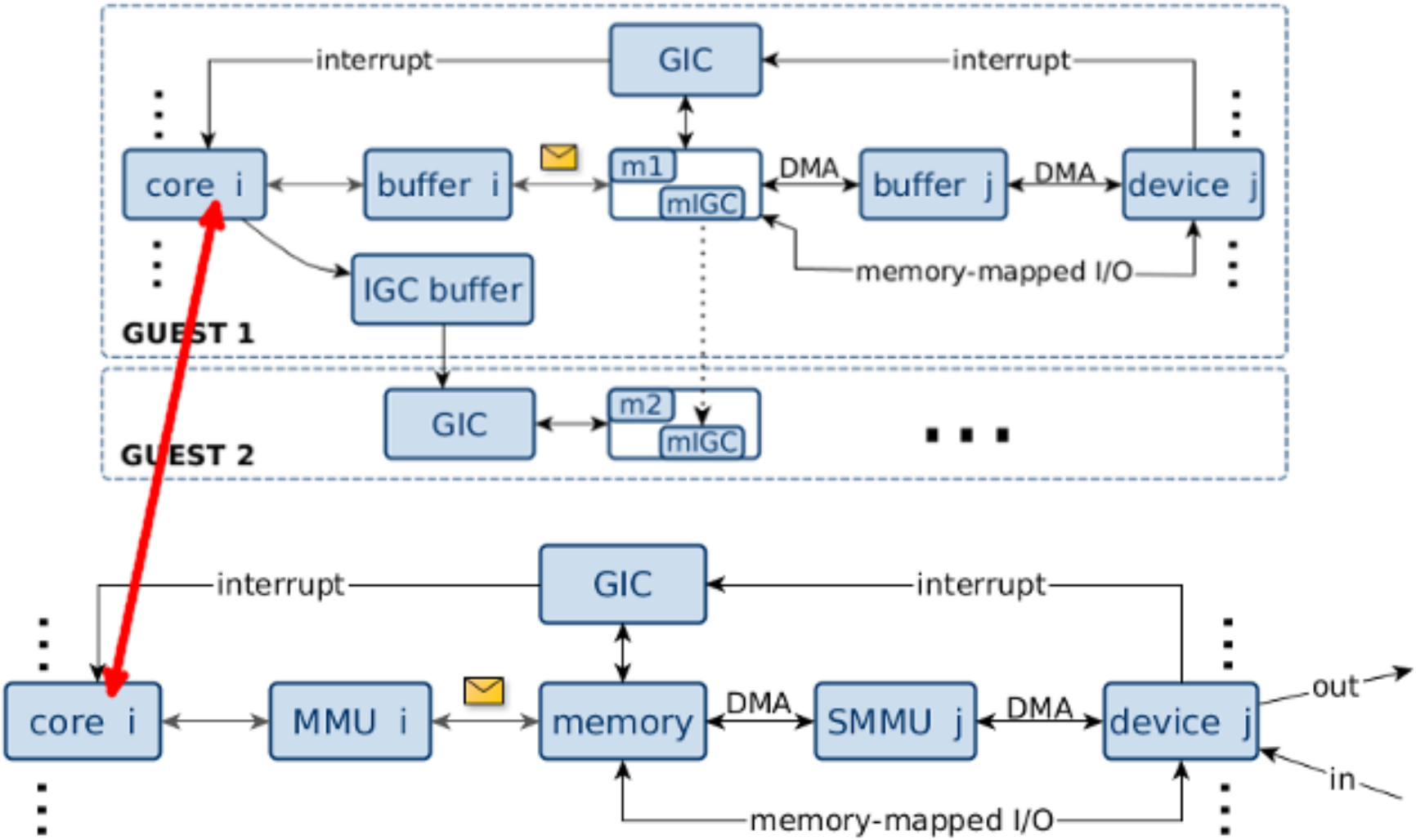
Proof by induction on transition sequence:

- for any initial state σ_P^0, $Inv(\sigma_P^0)$ and exists σ_I^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for any initial state σ_I^0, exists σ_P^0 such that $\sigma_P^0\ R\ \sigma_I^0$
- for σ_P and σ_I with $\sigma_P\ R\ \sigma_I$ and $Inv(\sigma_P)$:
  - $\sigma_P \longrightarrow \sigma_P' \implies \exists \sigma_I'.\ \sigma_I \longrightarrow^* \sigma_I' \ \wedge\ \sigma_P'\ R\ \sigma_I'$
  - $\sigma_I \longrightarrow \sigma_I' \implies \exists \sigma_P'.\ \sigma_P \longrightarrow^* \sigma_P' \ \wedge\ \sigma_P'\ R\ \sigma_I'$

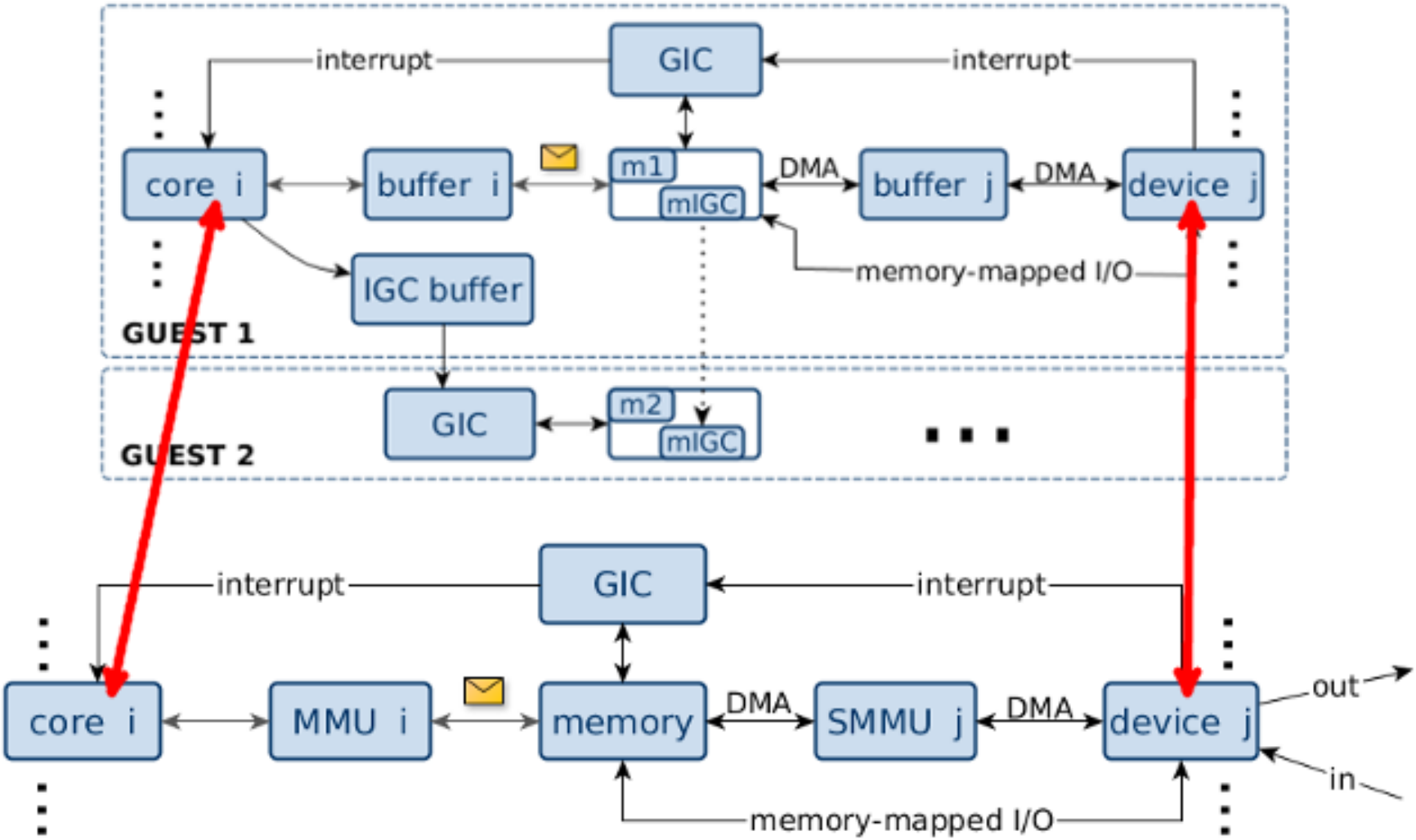
Verification: Bisimulation Relation



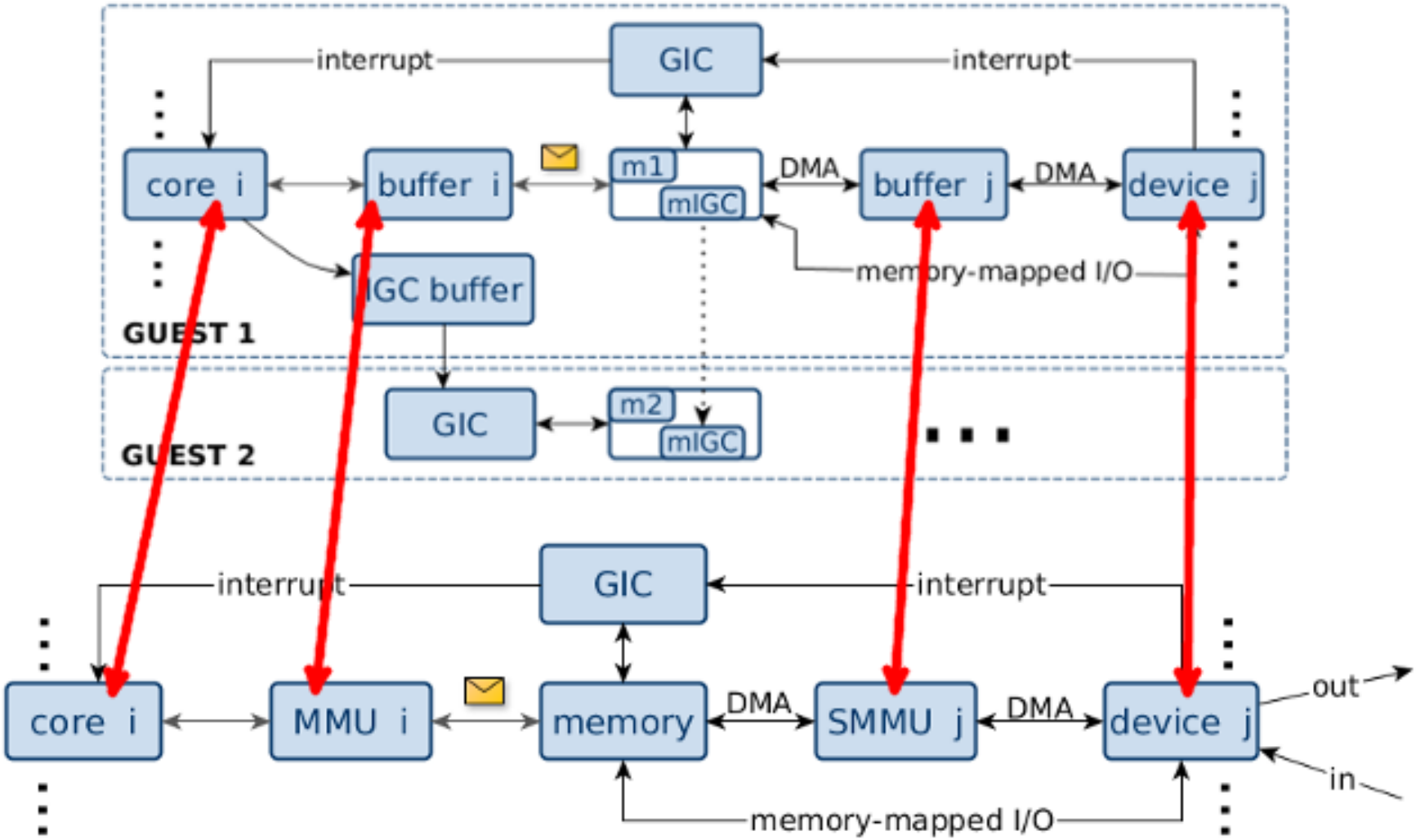
Verification: Bisimulation Relation



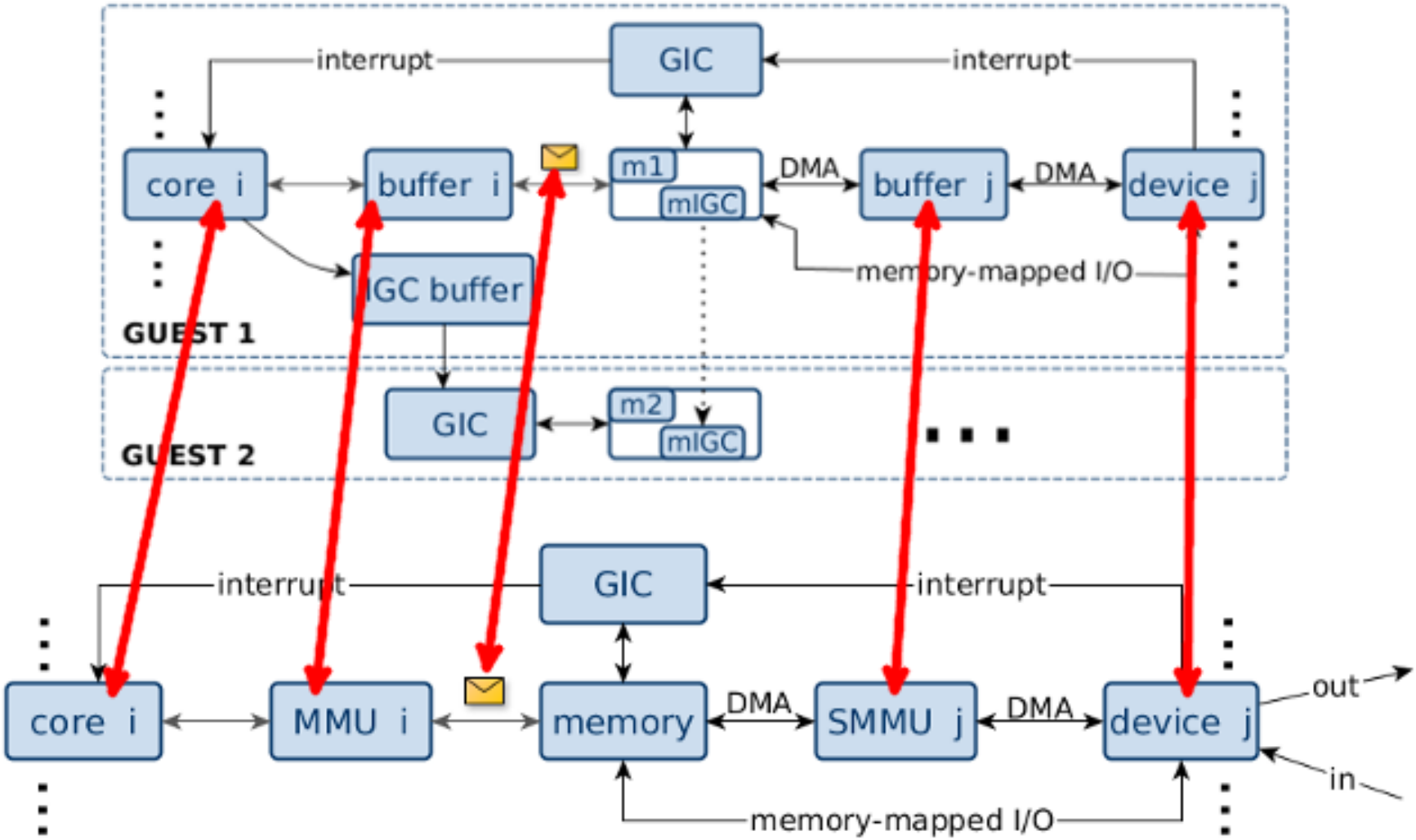
Verification: Bisimulation Relation



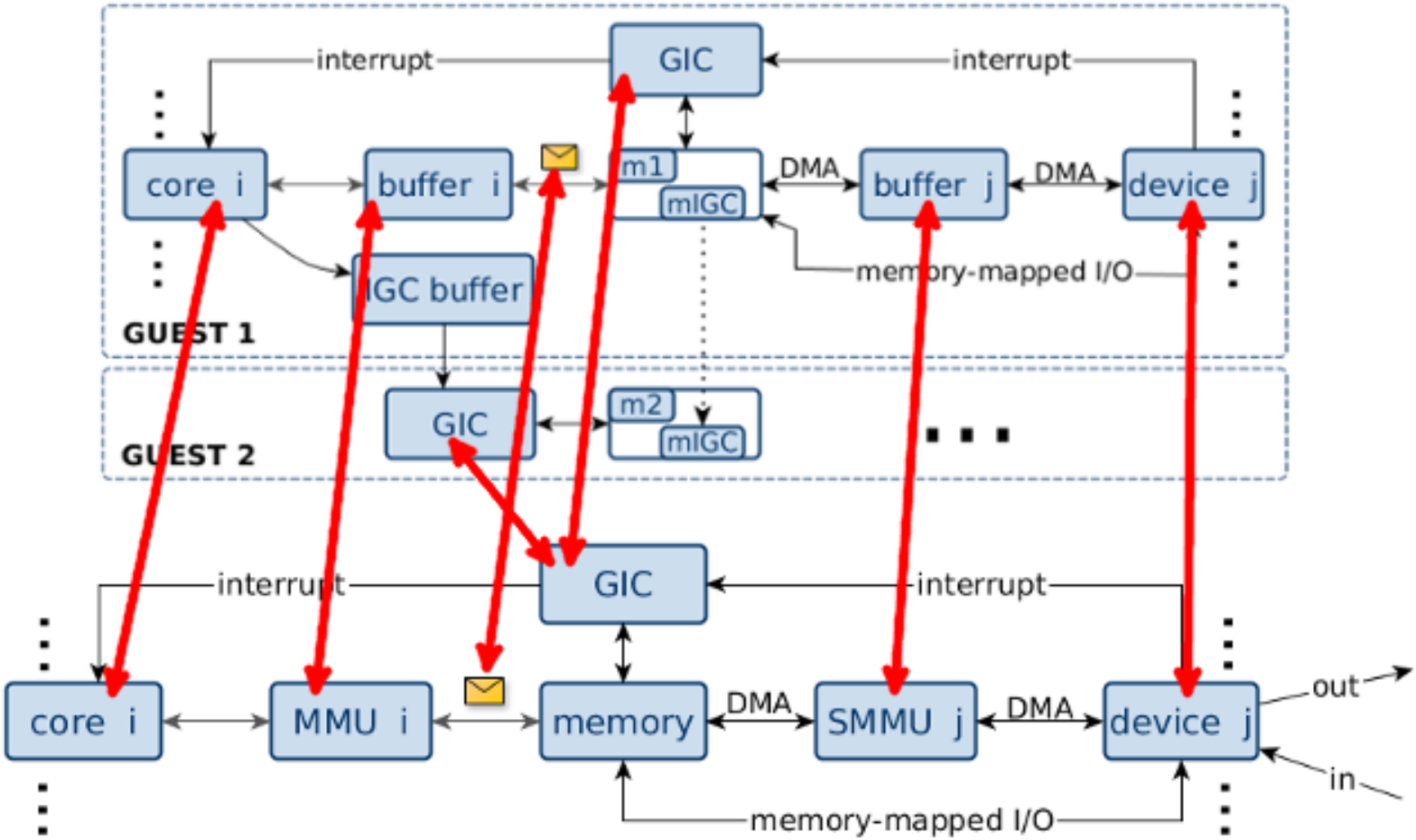
Verification: Bisimulation Relation



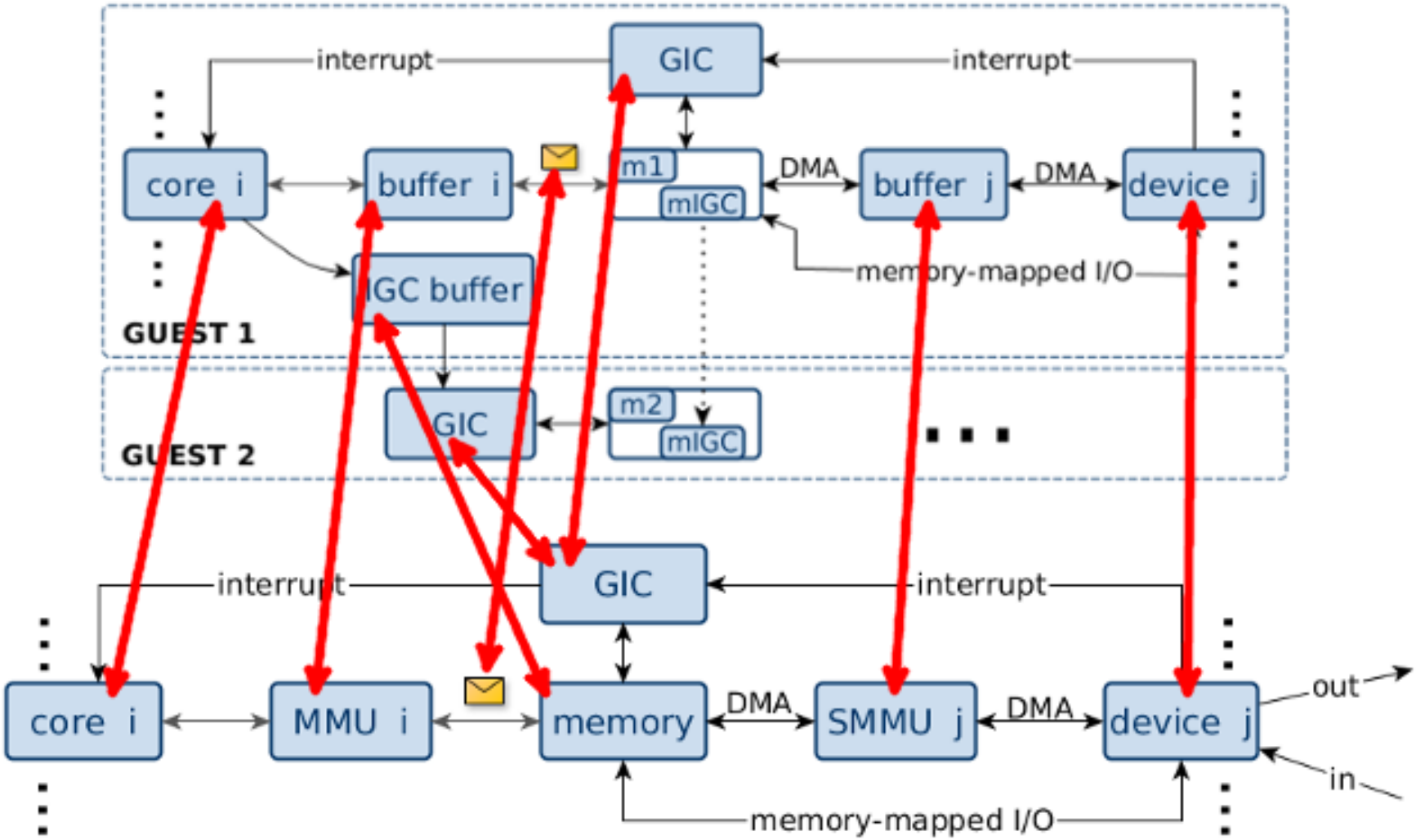
Verification: Bisimulation Relation



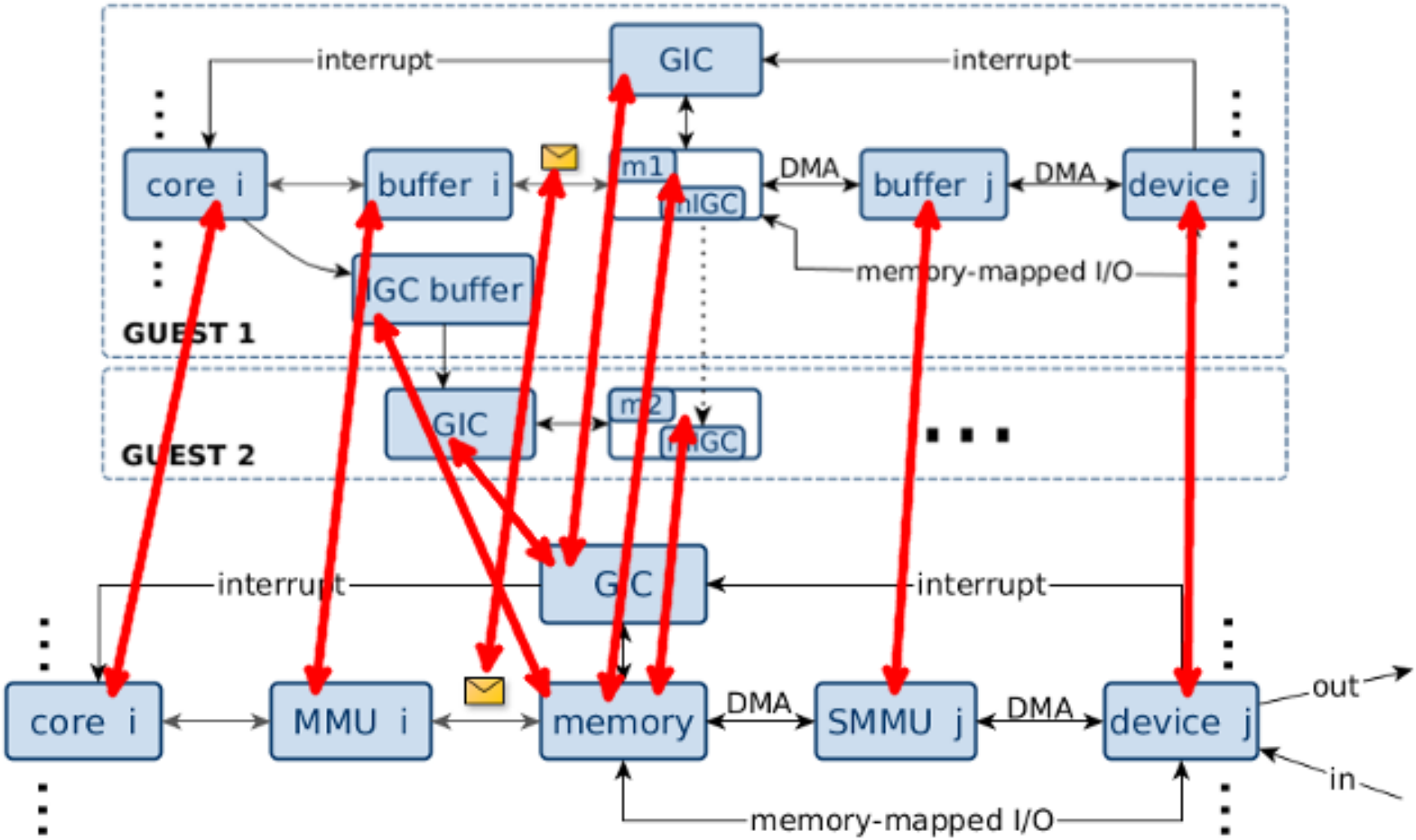
Verification: Bisimulation Relation



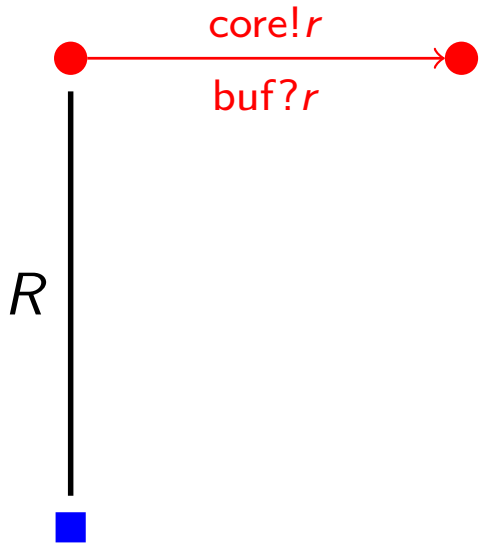
Verification: Bisimulation Relation



Verification: Bisimulation Relation

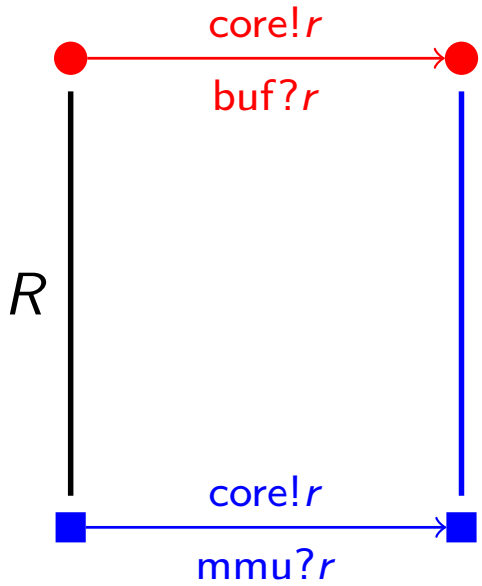


Verification: MMU steps



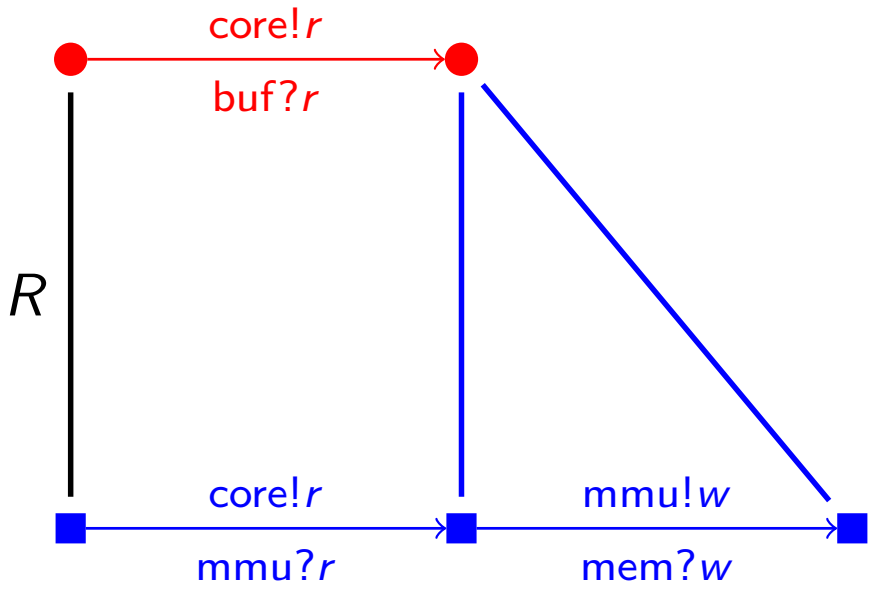
- ideal core sends memory request r to "MMU" buffer

Verification: MMU steps



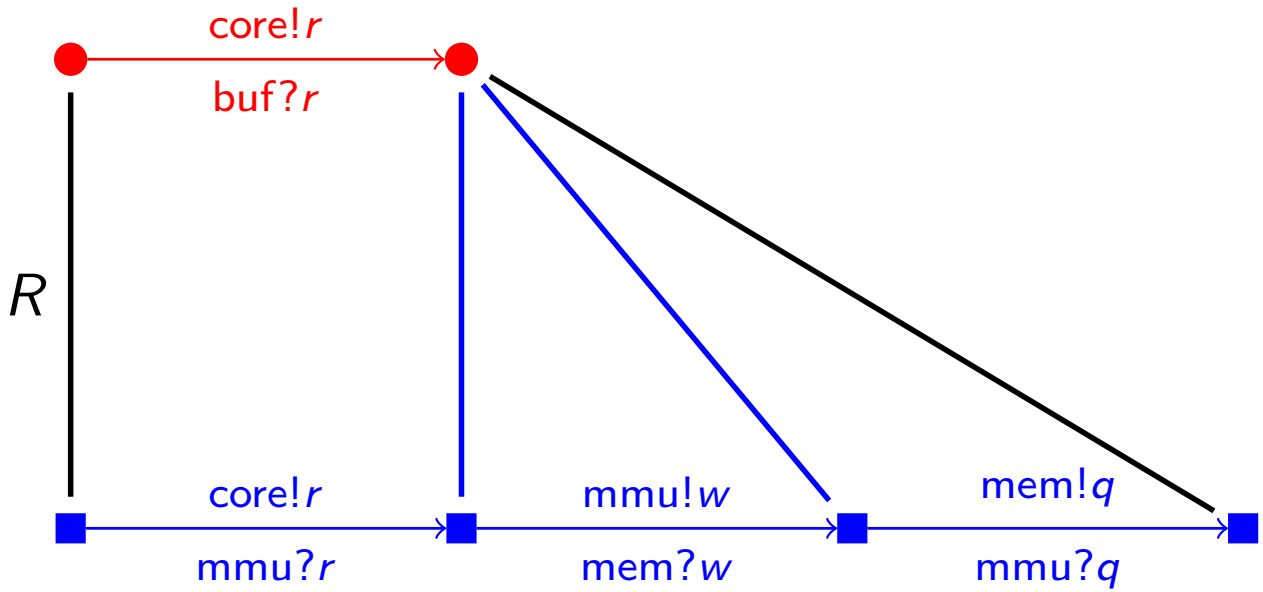
- ideal core sends memory request r to "MMU" buffer
- same request sent in platform model

Verification: MMU steps



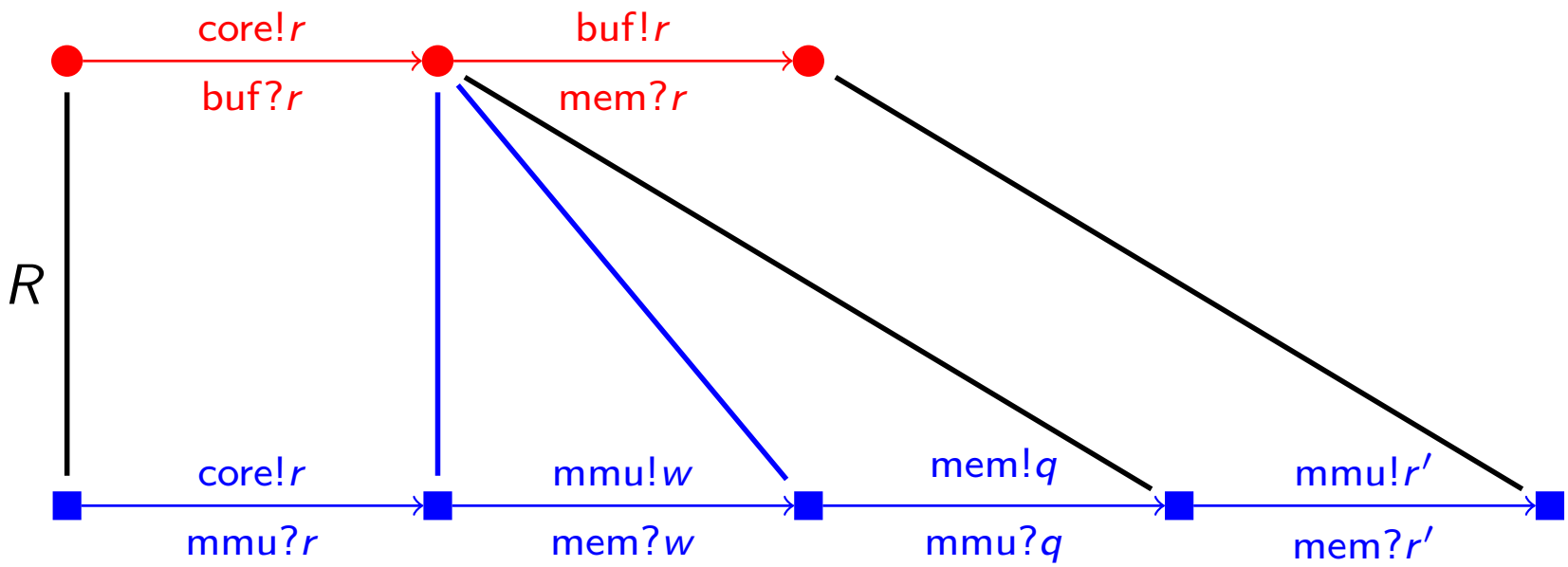
- ideal core sends memory request r to "MMU" buffer
- same request sent in platform model
- MMU sends page table lookup w to memory

Verification: MMU steps



- ideal core sends memory request r to "MMU" buffer
- same request sent in platform model
- MMU sends page table lookup w to memory
- memory answers with reply q, matching w, translation $r \mapsto r'$

Verification: MMU steps



- ideal core sends memory request r to "MMU" buffer
- same request sent in platform model
- MMU sends page table lookup w to memory
- memory answers with reply q, matching w, translation $r \mapsto r'$
- (translated) request forwarded to memory

Bisimulation Proof: Implementation

	basic	common	ideal	platform	hyperv.	bisim.	total
model specification	99	435	1,121	1,750	1,440	350	5,195
invariant specification	–	17	387	518	–	453	1,375
machinery	309	–	95	–	–	585	989
proofs	652	1,094	1,132	1,466	145	7,437	11,926
total	1,060	1,546	2,735	3,734	1,585	8,825	19,485

- implemented in HOL4 theorem prover

Bisimulation Proof: Implementation

	basic	common	ideal	platform	hyperv.	bisim.	total
model specification	99	435	1,121	1,750	1,440	350	5,195
invariant specification	–	17	387	518	–	453	1,375
machinery	309	–	95	–	–	585	989
proofs	652	1,094	1,132	1,466	145	7,437	11,926
total	1,060	1,546	2,735	3,734	1,585	8,825	19,485

- implemented in HOL4 theorem prover
- most important cases verified

Bisimulation Proof: Implementation

	basic	common	ideal	platform	hyperv.	bisim.	total
model specification	99	435	1,121	1,750	1,440	350	5,195
invariant specification	–	17	387	518	–	453	1,375
machinery	309	–	95	–	–	585	989
proofs	652	1,094	1,132	1,466	145	7,437	11,926
total	1,060	1,546	2,735	3,734	1,585	8,825	19,485

- implemented in HOL4 theorem prover
- most important cases verified
- first steps towards automation

Bisimulation Proof: Implementation

	basic	common	ideal	platform	hyperv.	bisim.	total
model specification	99	435	1,121	1,750	1,440	350	5,195
invariant specification	–	17	387	518	–	453	1,375
machinery	309	–	95	–	–	585	989
proofs	652	1,094	1,132	1,466	145	7,437	11,926
total	1,060	1,546	2,735	3,734	1,585	8,825	19,485

- implemented in HOL4 theorem prover
- most important cases verified
- first steps towards automation
- simplifier and resolution solvers for trivial cases

Bisimulation Proof: Implementation

	basic	common	ideal	platform	hyperv.	bisim.	total
model specification	99	435	1,121	1,750	1,440	350	5,195
invariant specification	–	17	387	518	–	453	1,375
machinery	309	–	95	–	–	585	989
proofs	652	1,094	1,132	1,466	145	7,437	11,926
total	1,060	1,546	2,735	3,734	1,585	8,825	19,485

- implemented in HOL4 theorem prover
- most important cases verified
- first steps towards automation
- simplifier and resolution solvers for trivial cases
- proofs robust against local changes

Bisimulation Proof: Implementation

	basic	common	ideal	platform	hyperv.	bisim.	total
model specification	99	435	1,121	1,750	1,440	350	5,195
invariant specification	–	17	387	518	–	453	1,375
machinery	309	–	95	–	–	585	989
proofs	652	1,094	1,132	1,466	145	7,437	11,926
total	1,060	1,546	2,735	3,734	1,585	8,825	19,485

- implemented in HOL4 theorem prover
- most important cases verified
- first steps towards automation
- simplifier and resolution solvers for trivial cases
- proofs robust against local changes
- lots of technical lemmas

Assumptions and Caveats

- flat memory model

Assumptions and Caveats

- flat memory model
- one core memory request at a time

Assumptions and Caveats

- flat memory model
- one core memory request at a time
- one SMMU per device

Assumptions and Caveats

- flat memory model
- one core memory request at a time
- one SMMU per device
- peripherals inactive at boot

Assumptions and Caveats

- flat memory model
- one core memory request at a time
- one SMMU per device
- peripherals inactive at boot
- GICv2 model

Conclusion & Future work

Summary:

- compositonal approach to SoC modeling for security verification
- reusability of models, adaptability of proofs
- top-down approach, abstraction and late refinement
- early identification of invariants and proof obligations
- case study in HOL4

Conclusion & Future work

Summary:

- compositonal approach to SoC modeling for security verification
- reusability of models, adaptability of proofs
- top-down approach, abstraction and late refinement
- early identification of invariants and proof obligations
- case study in HOL4

TODOs:

- generalized formal framework, DSLs
- more automation
- advanced hardware features
- refinement of components
- property transfer

THANKS!

prosper.sics.se haspoc.sics.se