RESH and D-RESH: Fault-tolerant replication on the basis of modern virtualisation technology

Hans P. Reiser
Franz J. Hauck

Distributed Systems Lab,
Faculty of Computer Science

Ulm University
Germany

Rüdiger Kapitza

Department of Distributed Systems
and Operating Systems

University of Erlangen-Nürnberg
Germany

Aspectix Research Group
http://www.aspectix.org/

13 October 2006

Roadmap

1. Background: Virtualisation and dependability research
2. Challenges in fault-tolerant systems
3. RESH and D-RESH: Our approaches to hypervisor-based fault tolerance
4. Conclusions

Roadmap

1. Background: Virtualisation and dependability research
   - Current virtualisation technology
   - System verification using formal methods
   - Fighting intrusions with virtualisation
   - Replication on the basis of virtualisation
2. Challenges in fault-tolerant systems
3. RESH and D-RESH: Our approaches to hypervisor-based fault tolerance
4. Conclusions

Background: Current virtualisation technology

Recently, virtualisation has re-appeared in operating system research

- Integrated into OS: VMware GSX server, User Mode Linux, FAUMachine
- Hypervisor layer below OS: Xen, VMware ESX server
- CPU support for virtualisation: Intel Vanderpool, AMD Pacifica

Basic idea: Provide multiple virtual machines, each executing isolated operating system instances

Background: Current virtualisation technology

Recently, virtualisation has re-appeared in operating system research

- Integrated into OS: VMware GSX server, User Mode Linux, FAUMachine
- Hypervisor layer below OS: Xen, VMware ESX server
- CPU support for virtualisation: Intel Vanderpool, AMD Pacifica

Basic idea: Provide multiple virtual machines, each executing isolated operating system instances

- Benefits not discussed here: optimisation of CPU usage, resource management, ...
- *Our focus: dependability*

Background: Encapsulation increases reliability

Hypervisor provides *isolation* between independent virtual machines

Example: LeVasseur et al. (OSDI'04): Improved system dependability with hypervisor-based encapsulation

Background: Encapsulation increases reliability

Hypervisor provides *isolation* between independent virtual machines

> *Example: LeVasseur et al. (OSDI'04): Improved system dependability with hypervisor-based encapsulation*

Using a *small hypervisor* and separating the system into *isolated small components* reduces code size of components to an amount that can be handled with today's *verification tools*.

Background: Encapsulation enables formal verification

Using virtualisation enables *formal verification* of individual components:

- Verification of low-level microkernels:
  - Hohmuth et al. (ECOOP-PLOS'05): The VFiasco approach for a verified operating system memory management of kernel;
  - Tuch et al. (HOT-OS'05): OS verification – now!
- Verification of higher-level components on microkernel
  Völp (WDES'06): Verification of L4.Sec system services
- Robin Project (http://robin.tudos.org): Small, robust platform that can undergo formal analysis

Background: Encapsulation enables formal verification

Using virtualisation enables *formal verification* of individual components:

- Verification of low-level microkernels:
  - Hohmuth et al. (ECOOP-PLOS'05): The VFiasco approach for a verified operating system memory management of kernel;
  - Tuch et al. (HOT-OS'05): OS verification – now!
- Verification of higher-level components on microkernel
  Völp (WDES'06): Verification of L4.Sec system services
- Robin Project (http://robin.tudos.org): Small, robust platform that can undergo formal analysis

Not feasible for full operating system or complex applications! And this will not change the next 10 years...

Background: Fighting intrusions with virtualisation

Several researchers focus on handling intrusions with virtualisation technology:

- Dunlap et al. (OSDI'02): Intrusion analysis through virtual-machine logging and replay
- Garfinkel&Rosenblum (NDSSS'03): Intrusion detection on the basis of virtual machine introspection
- Kiyanclar (CCGRID'06): Survey of virtualisation focusing on secure on-demand cluster computing

Background: Fighting intrusions with virtualisation

Several researchers focus on handling intrusions with virtualisation technology:

- Dunlap et al. (OSDI'02): Intrusion analysis through virtual-machine logging and replay
- Garfinkel&Rosenblum (NDSSS'03): Intrusion detection on the basis of virtual machine introspection
- Kiyanclar (CCGRID'06): Survey of virtualisation focusing on secure on-demand cluster computing

Bad guys can do the same: virtualisation-based viruses etc.

- Rutkowska (SyScan'06): Subverting Vista kernel for fun and profit ("Blue Pill")

Motivation: Replication with virtual machines

Hypervisors can also be used for replication:

- Bressoud&Schneider (ACM TOCS 14(1), 1996): Hypervisor-based fault tolerance

Low-level approach, virtual machines are synchronised by the hypervisor on a per-instruction basis.

Roadmap

1. Background: Virtualisation and dependability research
2. Challenges in fault-tolerant systems
   - Replicating existing applications
   - The crash-stop illusion
   - Systematic software faults
3. RESH and D-RESH: Our approaches to hypervisor-based fault tolerance
4. Conclusions

Challenges: Replicating existing applications

Reuse of existing applications: transparent replication

- Development of new devices/services usually starts without considering dependability or security

Challenges: Replicating existing applications

Reuse of existing applications: transparent replication

- Development of new devices/services usually starts without considering dependability or security
  - Home PC without any protection + Internet
    $\rightarrow$ viruses, Trojan horses, spam, ...

Challenges: Replicating existing applications

Reuse of existing applications: transparent replication

- Development of new devices/services usually starts without considering dependability or security
  - Home PC without any protection + Internet
    $\longrightarrow$ viruses, Trojan horses, spam, ...
  - Mobile phones have increased functionality
    $\longrightarrow$ bluetooth attacks, spam calls, ...

Challenges: Replicating existing applications

Reuse of existing applications: transparent replication

- Development of new devices/services usually starts without considering dependability or security
  - Home PC without any protection + Internet
    $\longrightarrow$ viruses, Trojan horses, spam, ...
  - Mobile phones have increased functionality
    $\longrightarrow$ bluetooth attacks, spam calls, ...
- Thus, it is desirable to provide reliability in a *transparent way* or with *only few modifications* to existing applications

Challenges: The crash-stop illusion

Most fault-tolerance infrastructures assume crash-stop failures.
Many examples have proven this to be wrong.

- Hardware can by arbitrarily faulty
- Most systems can be affected by malicious intruders
  (in the extremely unlikely case of absence of serious software bugs, there is still the option of social engineering)

Challenges: The crash-stop illusion

Most fault-tolerance infrastructures assume crash-stop failures.
Many examples have proven this to be wrong.

- Hardware can by arbitrarily faulty
- Most systems can be affected by malicious intruders
  (in the extremely unlikely case of absence of serious software bugs, there is still the option of social engineering)

  $\Longrightarrow$ increasing research in Byzantine fault-tolerant systems

Challenges: The crash-stop illusion

Most fault-tolerance infrastructures assume crash-stop failures.
Many examples have proven this to be wrong.

- Hardware can by arbitrarily faulty
- Most systems can be affected by malicious intruders
  (in the extremely unlikely case of absence of serious software bugs, there is still the option of social engineering)

  $\Longrightarrow$ increasing research in Byzantine fault-tolerant systems
- Software itself can be faulty (thus causing *systematic* faults)

Challenges: Systematic software faults

N-version-programming: avoiding systematic faults

- Systematic software faults will affect all replicas
- N version programming helps

Challenges: Systematic software faults

N-version-programming: avoiding systematic faults

- Systematic software faults will affect all replicas
- N version programming helps
- N version programming is becoming popular on standard platforms, reusing existing diverse implementations

  Example: Gashi et al. (DSN'04): Fault diversity among off-the-shelf SQL servers

Roadmap

1. Background: Virtualisation and dependability research
2. Challenges in fault-tolerant systems
3. RESH and D-RESH: Our approaches to hypervisor-based fault tolerance
   - RESH: Redundant Execution on a Single Host
   - D-RESH: Distributed RESH
4. Conclusions

Our approach: RESH and D-RESH

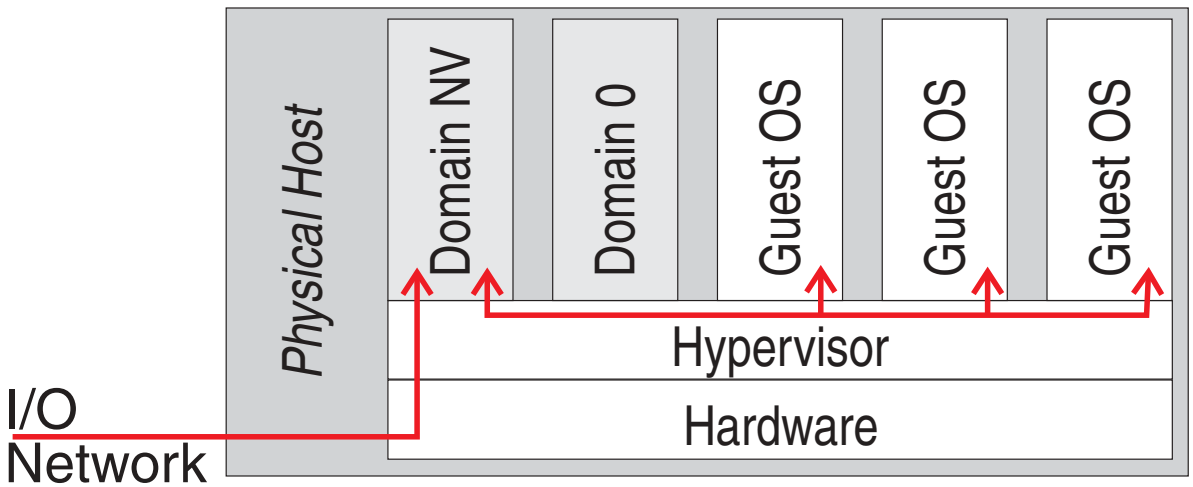
RESH: Redundant Execution on a Single Host

- Tolerating non-benign random faults on a single host
- N-version programming on a single host
- Low-level interception for transparent replication

D-RESH: Distributed RESH

- Small, trusted, verified "wormhole" for replication support
- Two-level replication: intra-host and inter-host
- Transparent hand-over on local networks

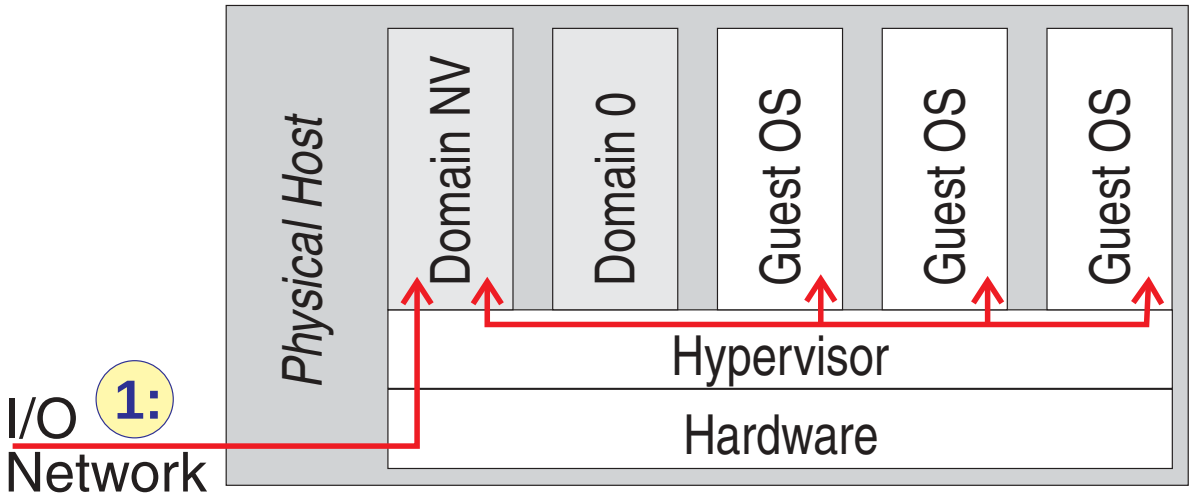
RESH: Basic Assumptions



- Network-based service (TCP or UDP interaction)
- A single, locally replicated service per machine

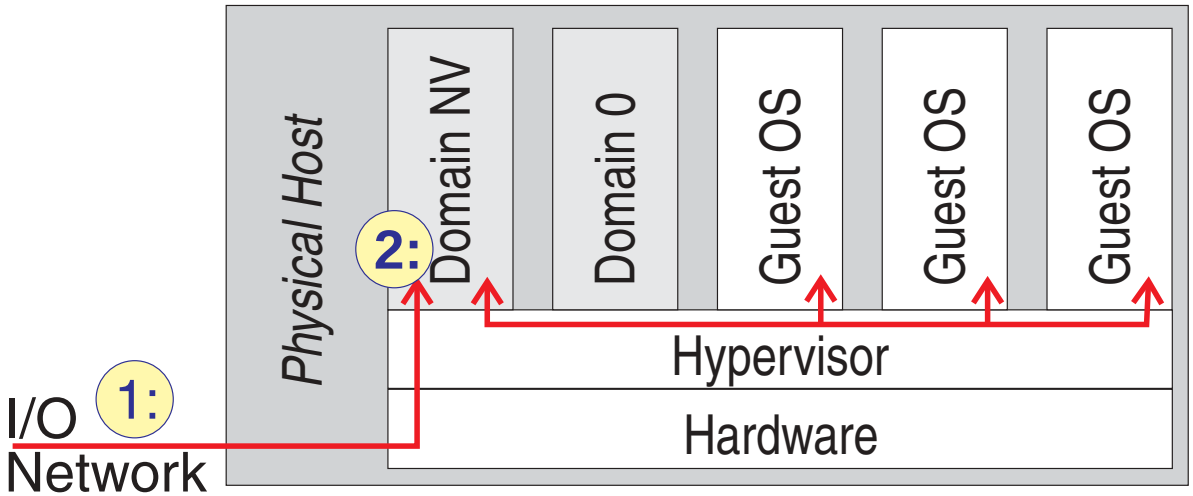
  Easily extended for multiple, potentially non-replicated, services as well

RESH architecture



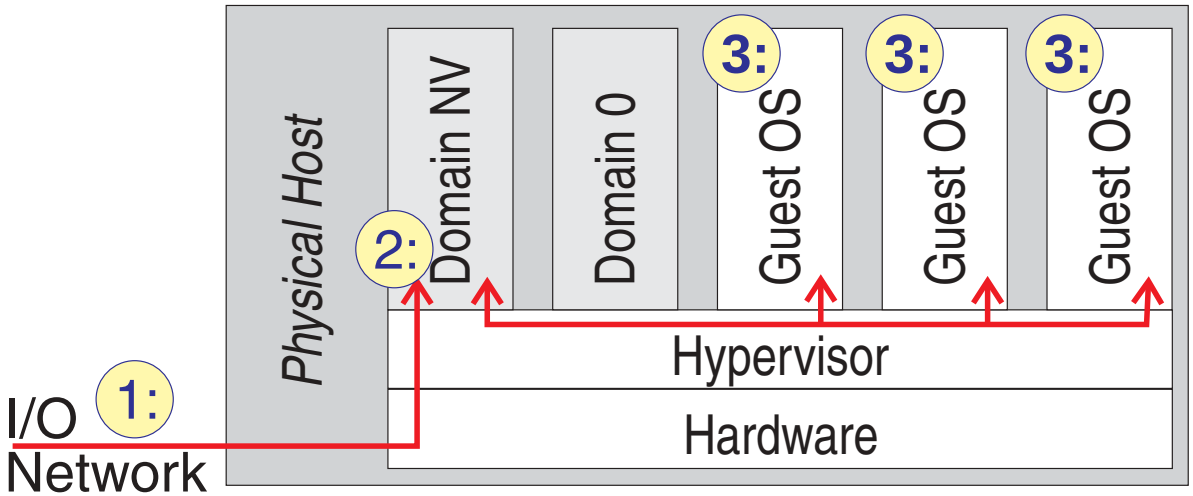
1. Client establishes connection and sends request

RESH architecture



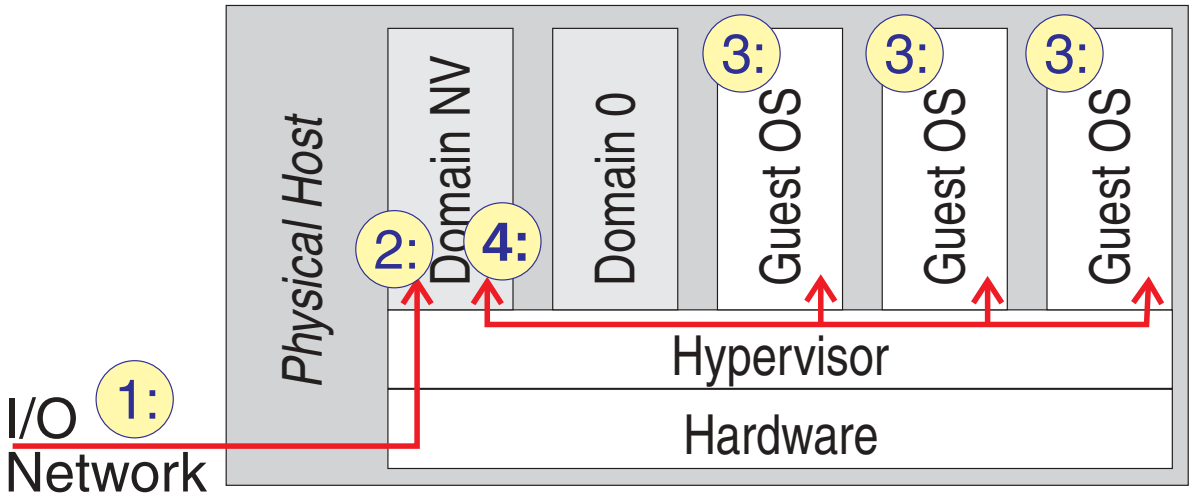
1. Client establishes connection and sends request
2. *Domain NV (network+voting)* forwards request to application (replicated in virtual machines)

RESH architecture



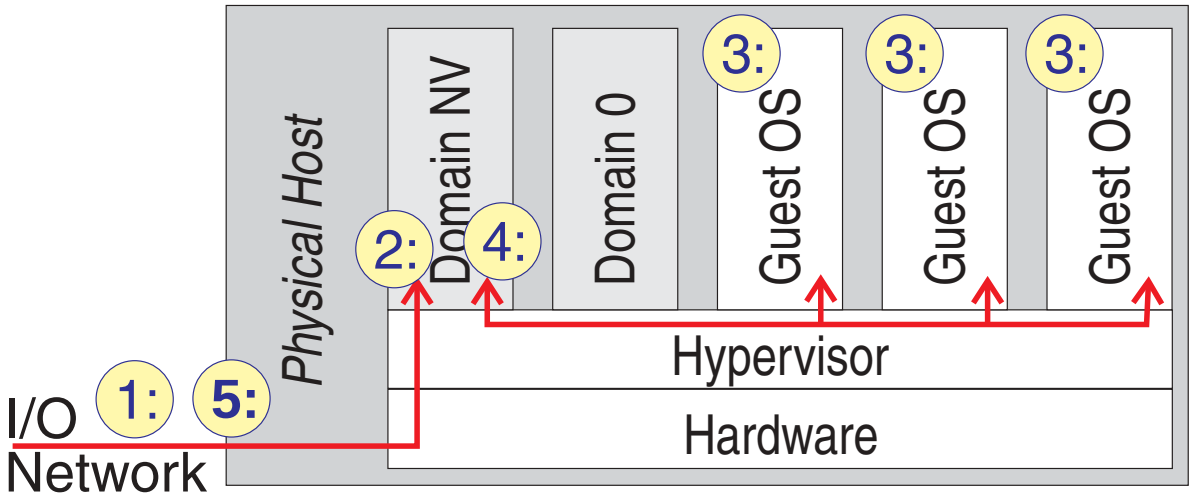
1. Client establishes connection and sends request
2. *Domain NV (network+voting)* forwards request to application (replicated in virtual machines)
3. Replicas execute client request and send result

RESH architecture



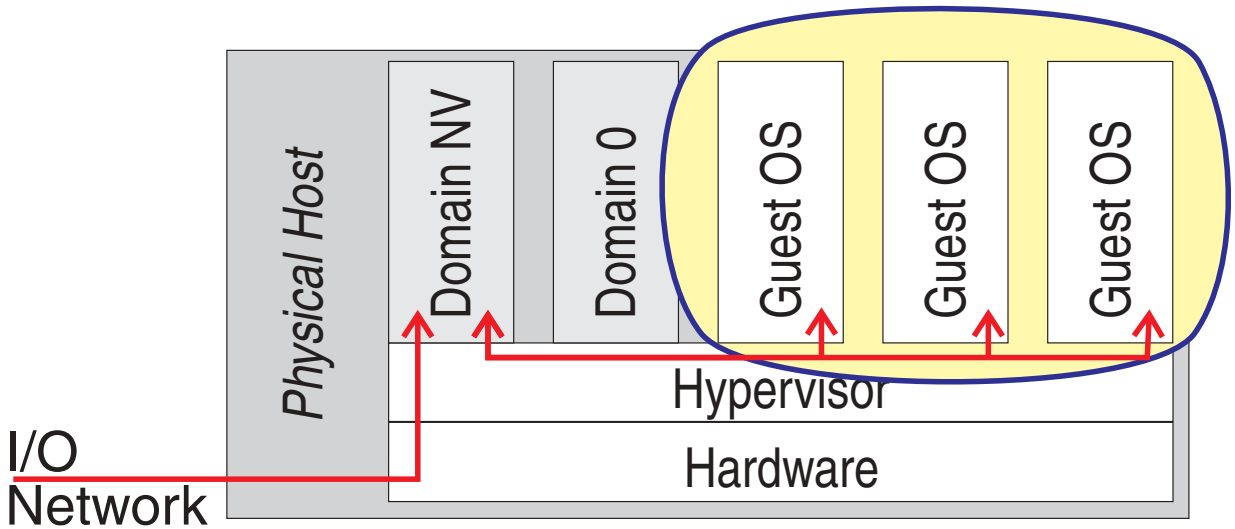
1. Client establishes connection and sends request
2. *Domain NV (network+voting)* forwards request to application (replicated in virtual machines)
3. Replicas execute client request and send result
4. *Domain NV* votes on return stream

RESH architecture



1. Client establishes connection and sends request
2. *Domain NV (network+voting)* forwards request to application (replicated in virtual machines)
3. Replicas execute client request and send result
4. *Domain NV* votes on return stream
5. *Domain NV* sends reply to client (after majority)

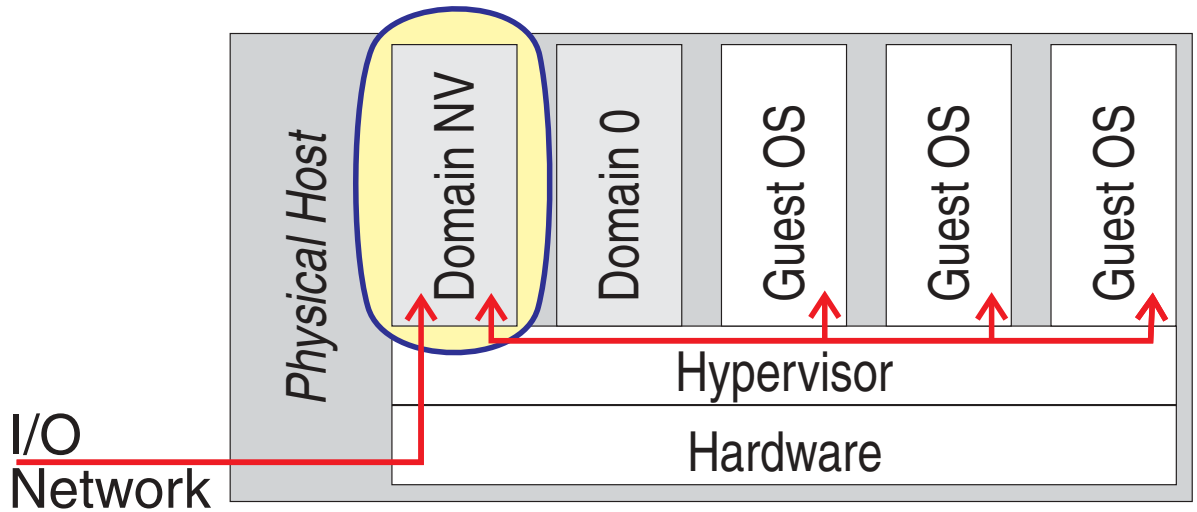
RESH: Service diversity



Service diversity of replicated virtual machines:

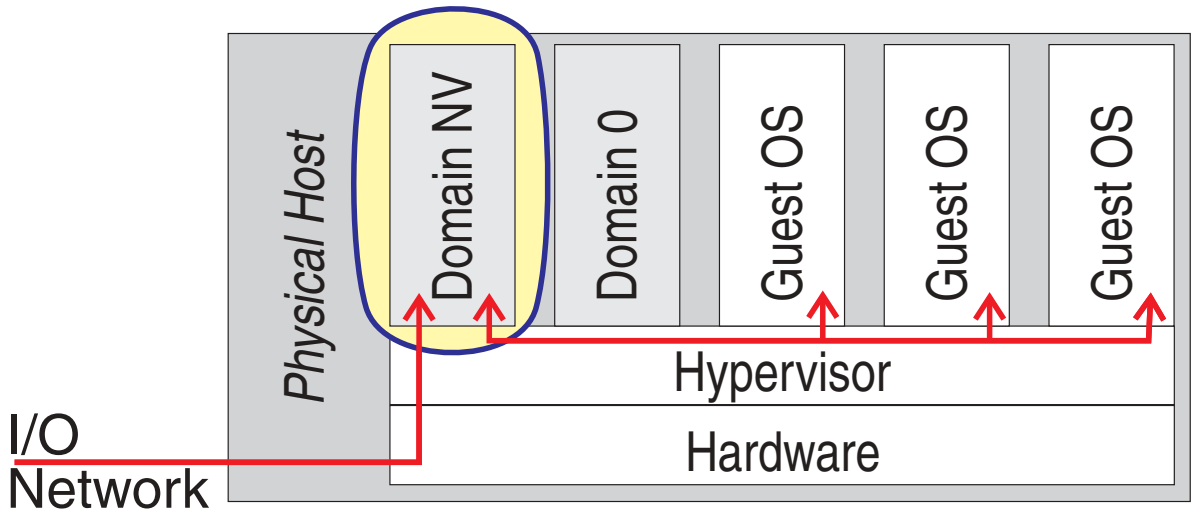
- OS diversity
- Middleware diversity
- Application diversity

RESH: Domain NV



- Hypervisor and Domain NV are a secure trusted computing base
- Small components: Formal verification feasible

RESH: Domain NV



- Hypervisor and Domain NV are a secure trusted computing base
- Small components: Formal verification feasible
- Tasks of Domain NV:
  - External communication: network device, TCP/IP stack
  - Local multicast to application virtual machines
  - Voting on replies

RESH architecture: configuration variants

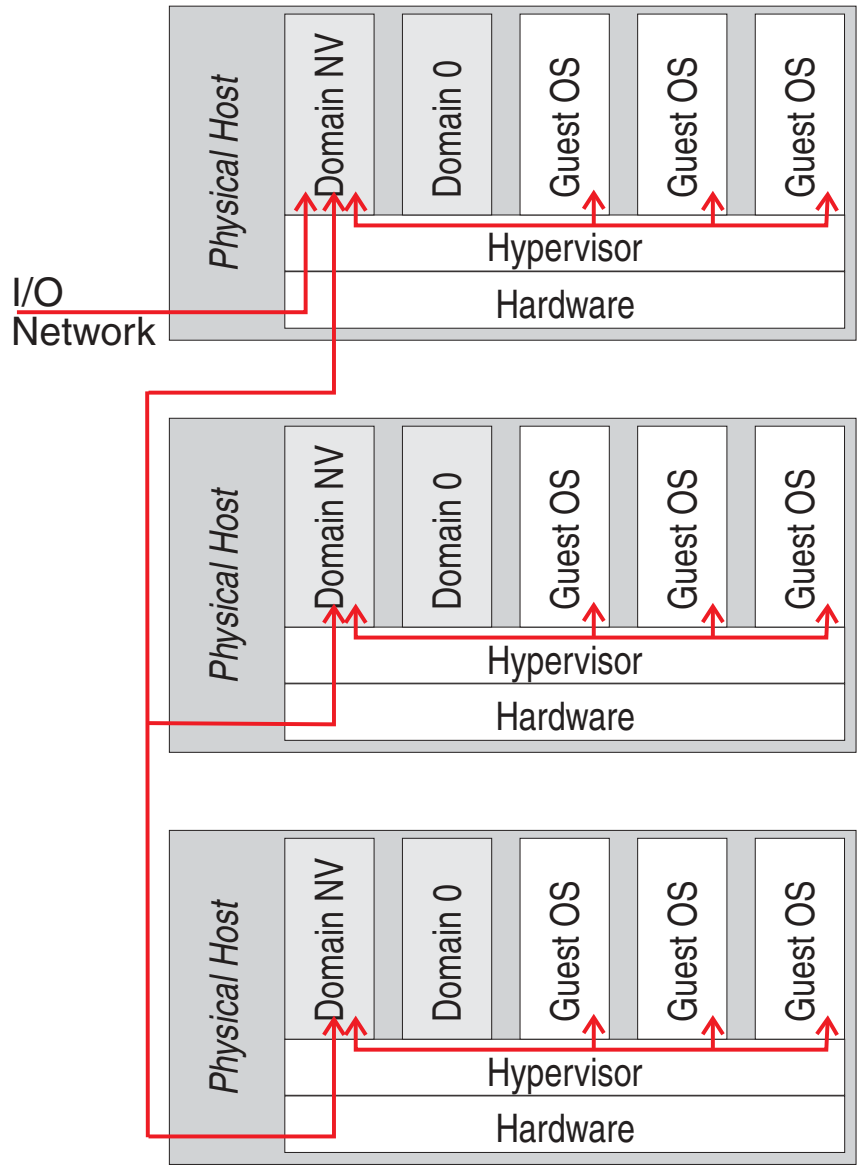
- Dual redundancy:
  Detect inconsistency and shut down system
  $\longrightarrow$ guarantees crash-stop behaviour
- Triple redundancy:
  Voting enables service continuation after failure; recovery of faulty virtual machine

RESH architecture: configuration variants

- Dual redundancy:
  Detect inconsistency and shut down system
  $\longrightarrow$ guarantees crash-stop behaviour
- Triple redundancy:
  Voting enables service continuation after failure; recovery of faulty virtual machine
- Extensions for distribution on multiple hosts: D-RESH

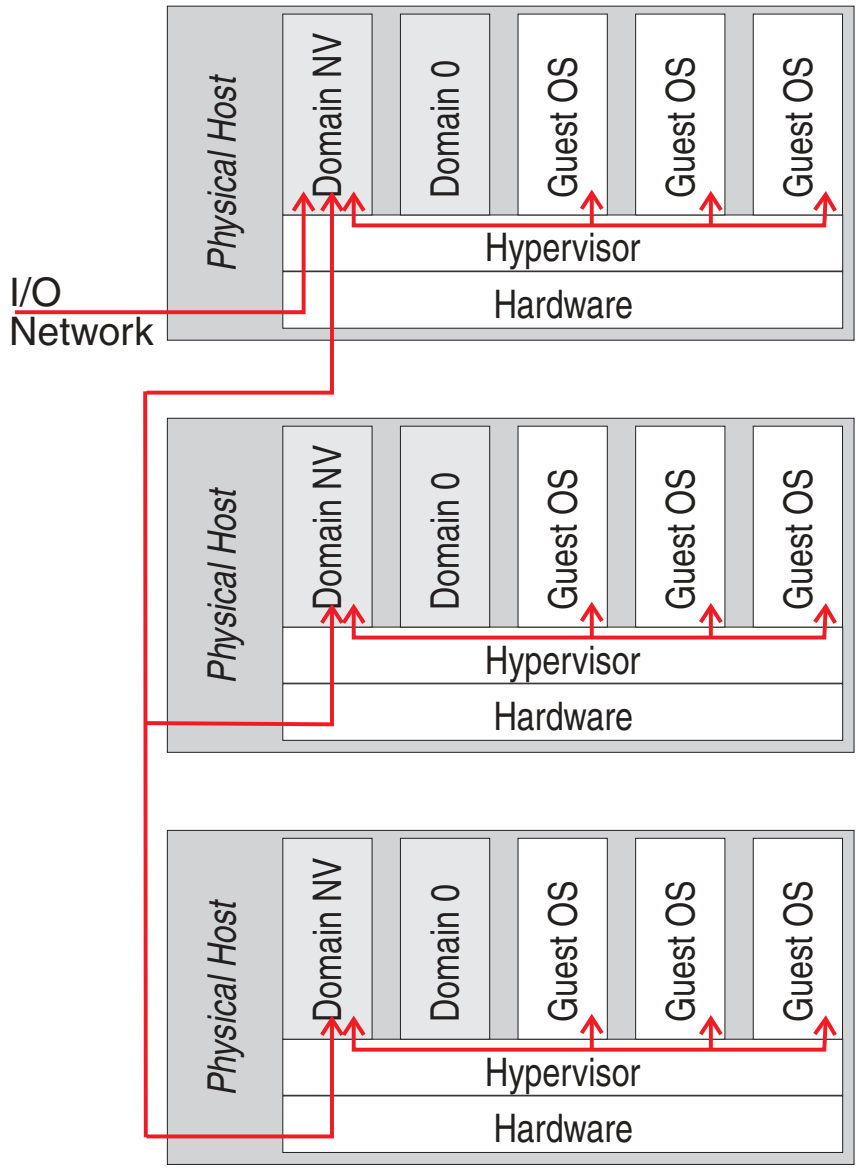
D-RESH: Replicating across host boundaries

- Use *Domain NV* for cross-host replication
  - Transparent interception of client requests
  - Trused "wormhole": no malicious intrusion at Hypervisor/Domain NV



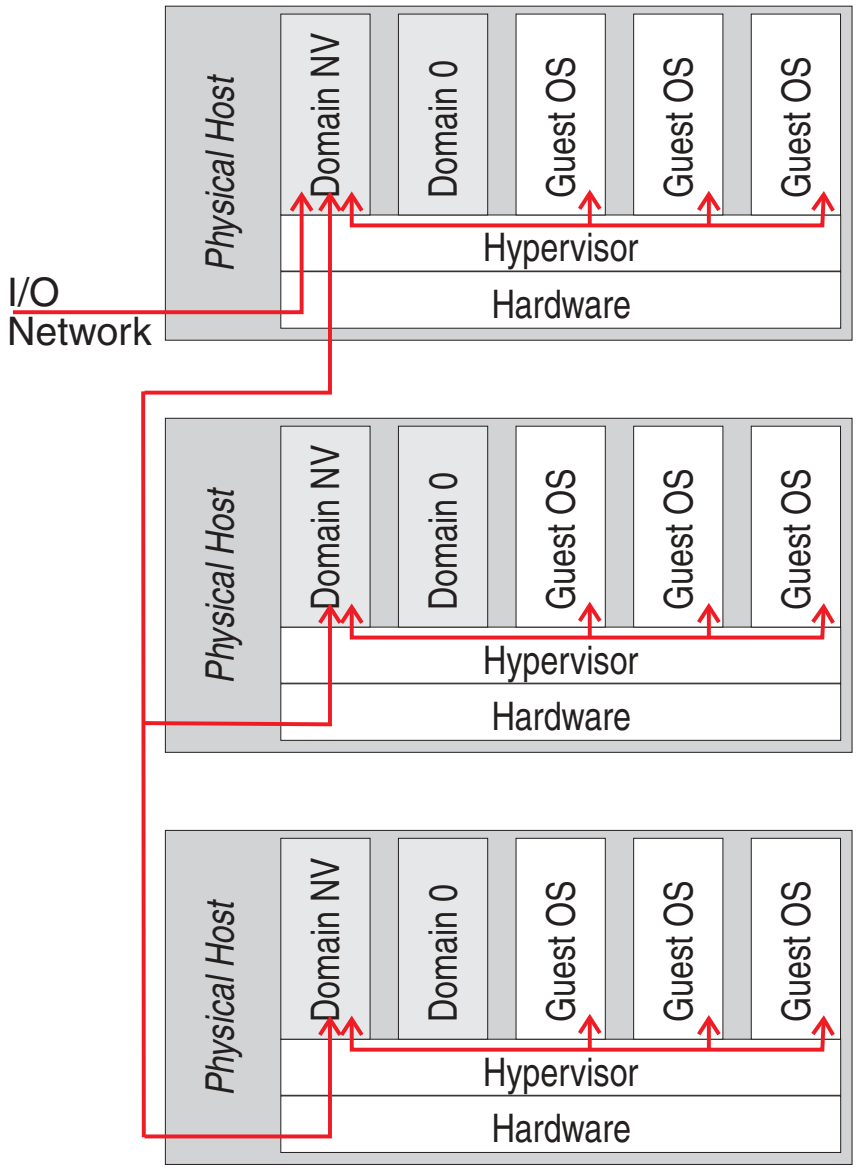
D-RESH: Replicating across host boundaries

- Use *Domain NV* for cross-host replication
  - Transparent interception of client requests
  - Trused "wormhole": no malicious intrusion at Hypervisor/Domain NV
- Integration of group communication



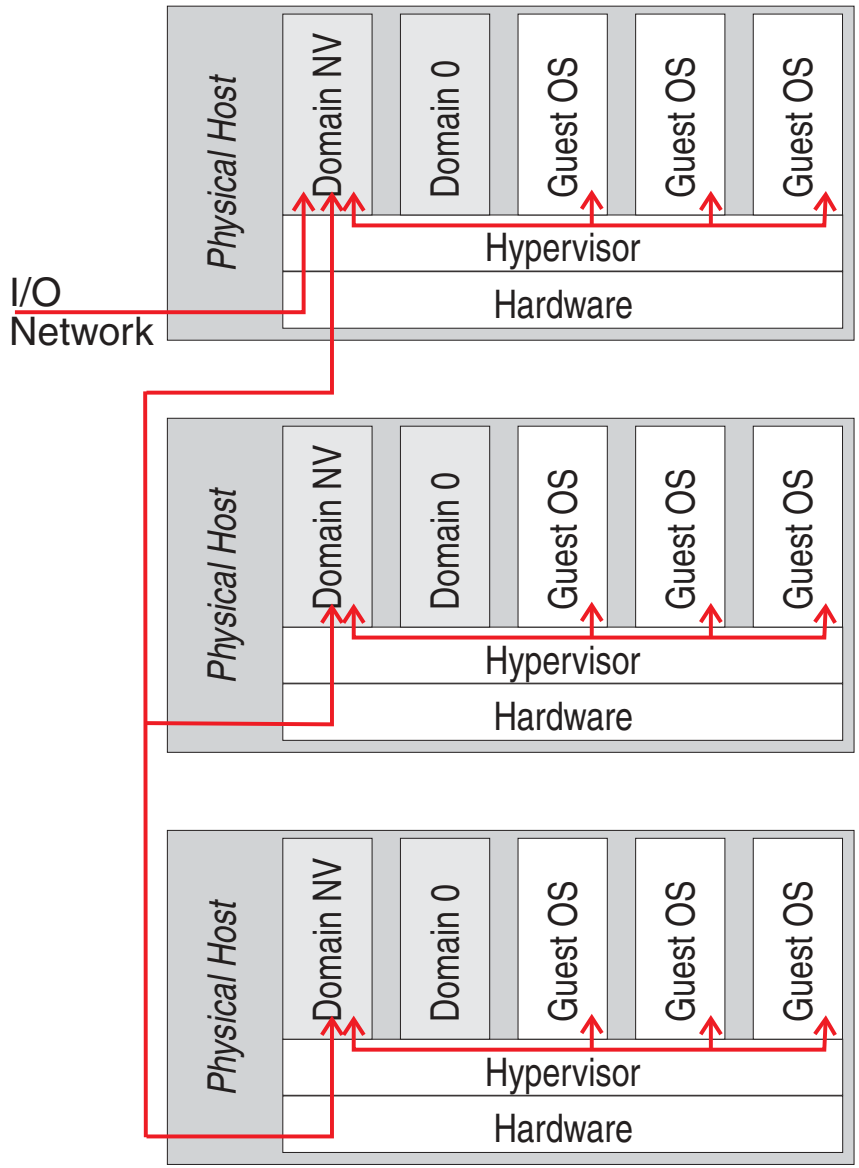
D-RESH: Replicating across host boundaries

- Use *Domain NV* for cross-host replication
  - Transparent interception of client requests
  - Trused "wormhole": no malicious intrusion at Hypervisor/Domain NV
- Integration of group communication
- TCP fail-over



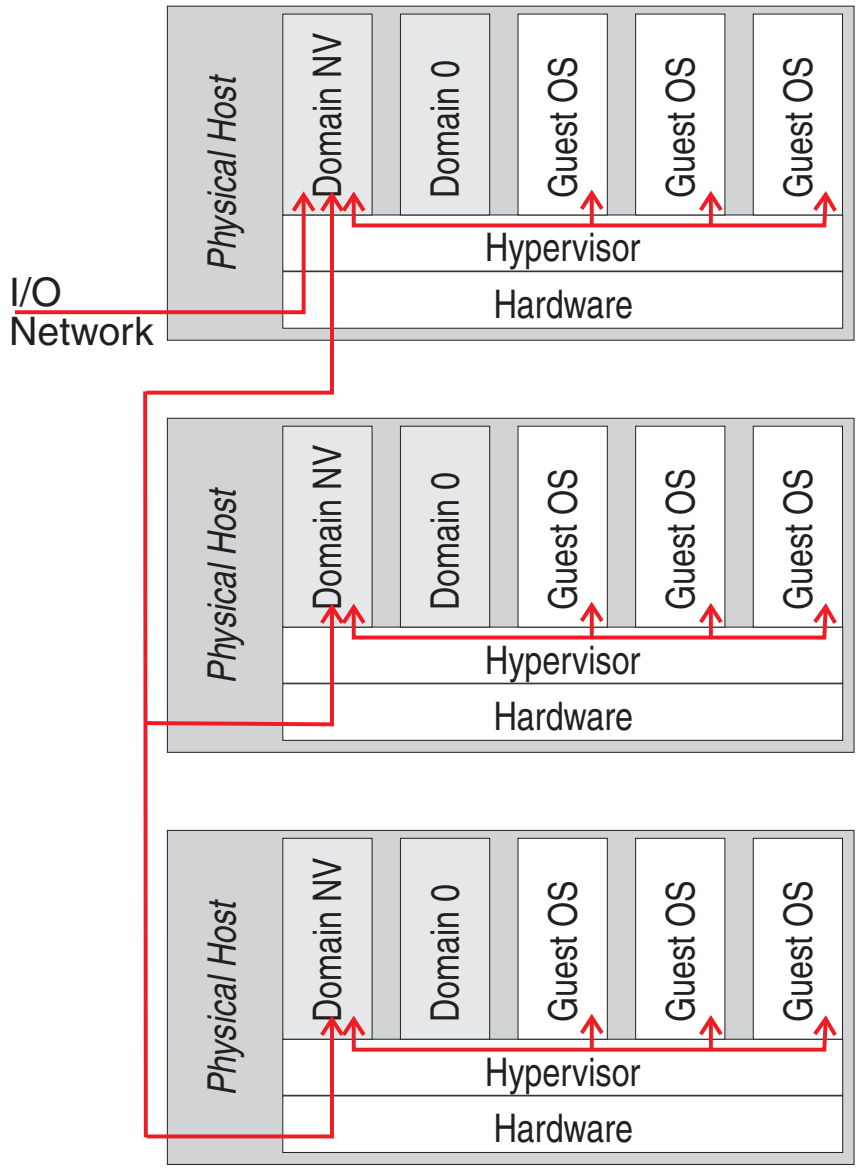
D-RESH: Replicating across host boundaries

- Use *Domain NV* for cross-host replication
  - Transparent interception of client requests
  - Trused "wormhole": no malicious intrusion at Hypervisor/Domain NV
- Integration of group communication
- TCP fail-over
  - Migrate IP address after failure



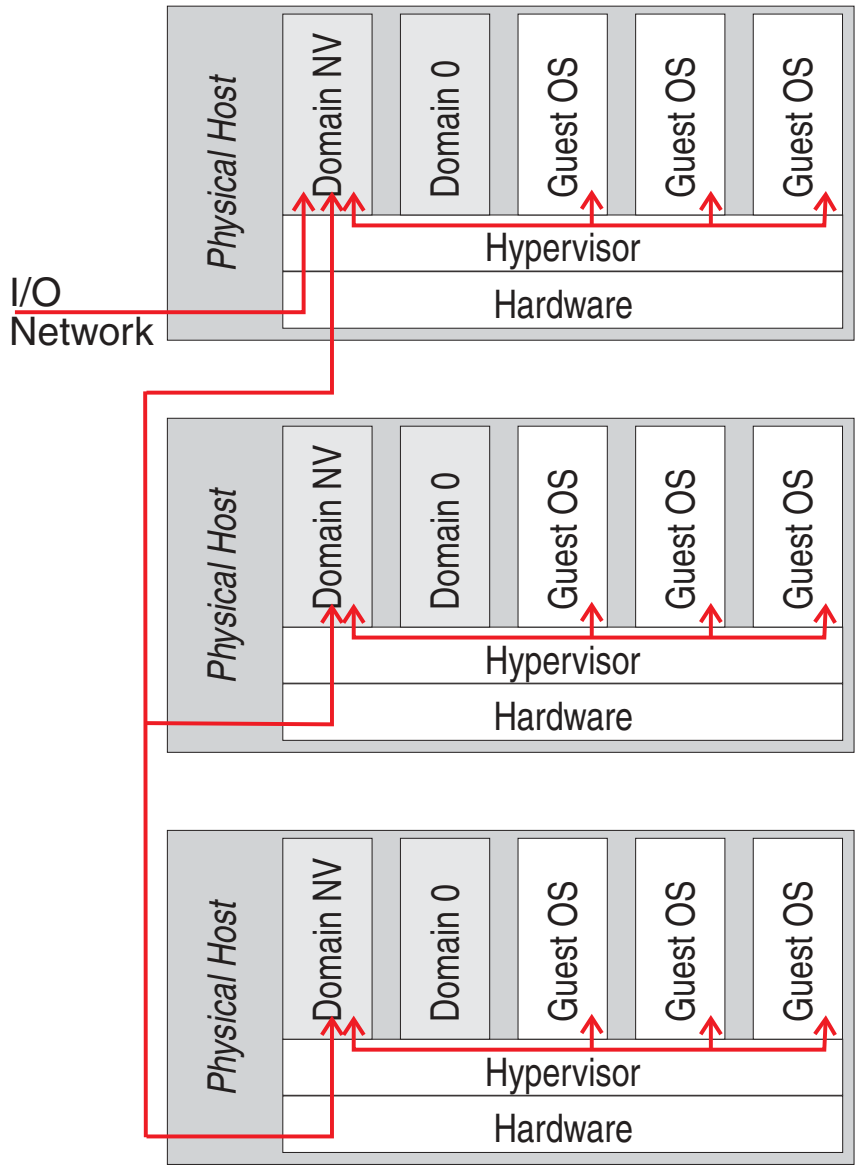
D-RESH: Replicating across host boundaries

- Use *Domain NV* for cross-host replication
  - Transparent interception of client requests
  - Trused "wormhole": no malicious intrusion at Hypervisor/Domain NV
- Integration of group communication
- TCP fail-over
  - Migrate IP address after failure
  - Migrate live IP stack



D-RESH: Replicating across host boundaries

- Use *Domain NV* for cross-host replication
  - Transparent interception of client requests
  - Trused "wormhole": no malicious intrusion at Hypervisor/Domain NV
- Integration of group communication
- TCP fail-over
  - Migrate IP address after failure
  - Migrate live IP stack
  - New transport-layer protocols?



Conclusions

- Virtualisation provides means for
  - Formal verifiability (small, isolated components)
  - Multiple virtual instances on a single machine
- RESH: Locally redundant execution using virtualisation
  - Isolation between redundant executions
  - Tolerating random non-crash faults
  - N-version programming
  - Efficient use of multi-core CPUs
- D-RESH: Distributed replication on multiple RESH hosts.
  - Transparent replication of existing applications
  - Trusted computing base on all nodes
  - Integration of group communication into Domain NV
  - TCP/IP failover