Verified secure kernels and hypervisors for the Cloud

Matthieu Lemerre Nikolai Kosmatov Céline Alec

CEA LIST

Séminaire INRIA Rennes, le 8 mars 2013

Plan

The need for isolation

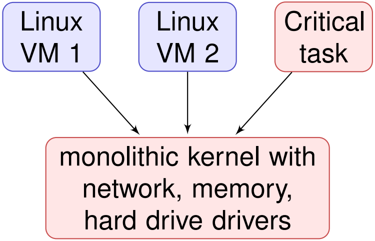
- The cloud mutualizes ressources (CPU time, memory, network bandwidth...) between tasks of several clients

➜ Often, each single computer in the cloud is shared

➜ Isolation between the tasks
  - Prevent a task from altering the behavior of another task (isolation)
  - Dually, prevent information from being accessed, modified, or made unavailable (information security/integrity and confidentiality)
- Anaxagoros:
  - Aims to provide the same level of isolation as physical separation
  - Allows secure, but dynamic and efficient ressource sharing
  - Favors reusability/ease of use through virtualization

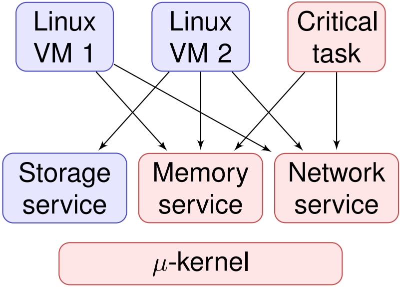
Anaxagoros: Design principles

- Intensive TCB minimization



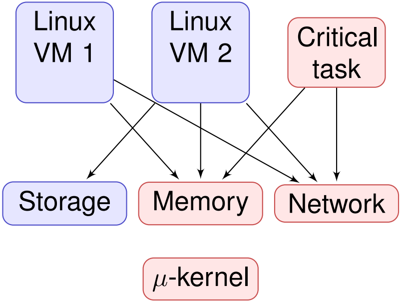
Anaxagoros: Design principles

- Intensive TCB minimization
  - Moving code and data from kernel to *isolated* services
    - Microkernel approach
    - Reduce impact of a fault to users of a service



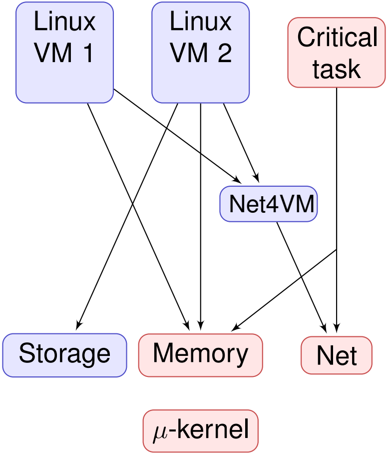
Anaxagoros: Design principles

- Intensive TCB minimization
  - Moving code and data from kernel to *isolated* services
    - Microkernel approach
    - Reduce impact of a fault to users of a service
  - Moving code and data from services to libraries in the tasks
    - Exokernel/hypervisor approach
    - Reduce impact of a fault to one task



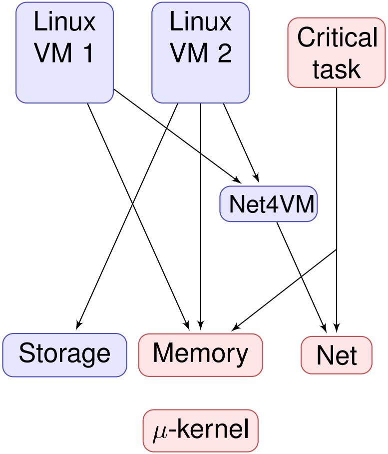
Anaxagoros: Design principles

- Intensive TCB minimization
  - Moving code and data from kernel to *isolated* services
    - Microkernel approach
    - Reduce impact of a fault to users of a service
  - Moving code and data from services to libraries in the tasks
    - Exokernel/hypervisor approach
    - Reduce impact of a fault to one task
  - Hierarchical ressource allocation and services
    - Move code from root to leafs
    - Reduce impact of a fault to users of the leaf service
  - Minimal impact of a fault or attack
  - Most trusted parts (kernel and root services) are smaller and isolated
  - ➜ Amenable to formal verification



Anaxagoros: Design principles

- Fast and precise access control
  - Unique, simple mechanism for access control: capabilities (keys)
  - Formalizes the access control links:
    - ➜ Analysis of the impact of the failure of a service (= tasks that use it)
    - ➜ Analysis of the vulnerabilities of a task (= used services + μ-kernel)
    - ➜ Simplified proof of isolation (reduced to shared services)
  - Behavioral isolation of a system is reduced to isolation of a small number of services
  - Innovative implementation:
    - all operations take $O(1)$ CPU time
    - capabilities take $O(1)$ kernel and service memory



Resource security: motivation

- Original motivations
  - Anaxagoros originally built for mixed-criticality hard real-time systems
  - Non-critical tasks must not slow down critical tasks
  - ➜ Protection against denial of services insufficient
  - Must protect against slow down of services
- Causes of task slow down
  - Hardware causes: cache evictions, bus contentions
  - Software causes: preventing execution of the highest priority task
    - Unpredictable blocking on semaphores, priority inversion
    - Priority inheritance
    - Exhausted resource (e.g. memory)
  - Usual solution: over-provisionning using pessimistic assumptions
    - E.g. static scheduling and allocations
    - Schedulability analysis with priority inheritance
- An alternative solution: “predictable” scheduling
  - Scheduler is always able to execute the task it wants to elect
  - \+ Better scheduling algorithms
  - \+ Less pessimistic schedulability analysis

Resource security: implementation

New resource security principle:

Independence of allocation policies

Allocation is defined in a single, separated module

- Applications
  - Allows to state and formally prove properties on resource allocation
    - Allows sharing resources (network, CPU time, memory) with *exact* accounting ($\rightarrow$ Cloud: billing)
    - (Provably) guaranteed QoS/performance isolation; critical real-time tasks
  - Security: allows suppression of resource-related covert channel
  - Allocation becomes a separate concern $\rightarrow$ modular design, custom allocation policies
- Requires to eliminate usual “ad-hoc” design decisions, e.g.:
  - Kernel that bypass the resource allocation module
  - Using blocking locks and semaphores in the OS
  - Denial of resources (hard, especially with isolated shared services)

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client

1. Denial of resource problem

Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client

1. Denial of resource problem

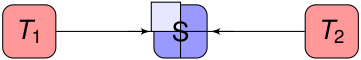
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

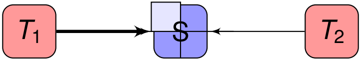
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

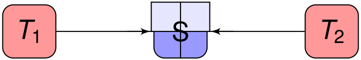
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

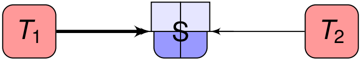
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

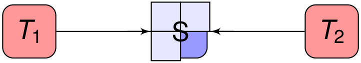
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

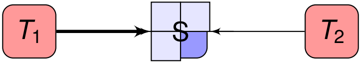
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

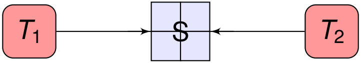
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

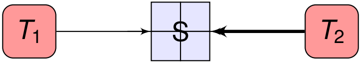
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

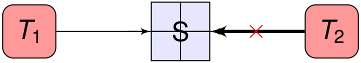
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - ➜ Client sends requests to services
  - Handling requests consume resources
  - ➜ Service consumes resources on behalf of the client



1. Denial of resource problem

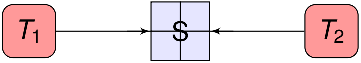
Denial of resource

Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

Resources when using a service

- Security put service and clients into separated protection domains
  - → Client sends requests to services
  - Handling requests consume resources
  - → Service consumes resources on behalf of the client



1. Denial of resource problem

Denial of resource

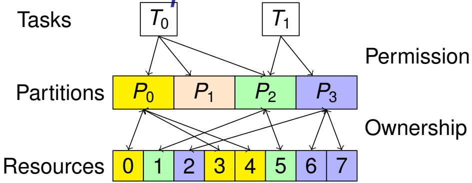
Sending requests that exhaust the resources of a service

- Ex: sending requests to an X server
  - Spend CPU time to execute the request
  - Spend memory to store queued requests

2. Resource accounting problem
   - How to attribute these resources to the client?

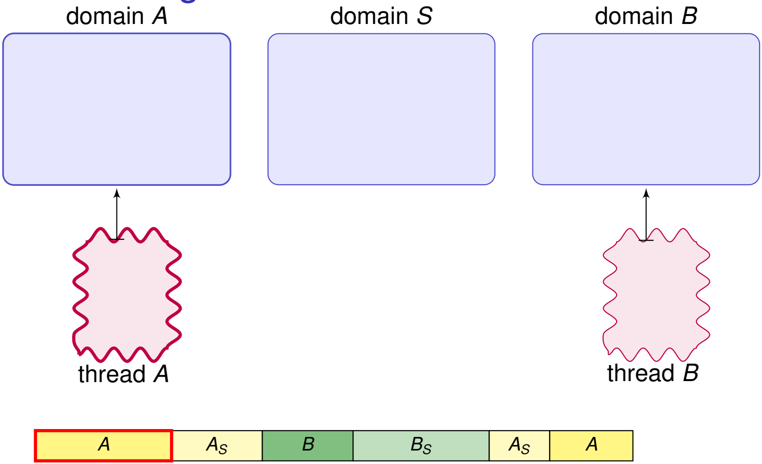
→ Our solution: complete *resource lending*

Separation between *permission* and *ownership*



- How to account for resources used by a task?
  - In a dynamic system (reallocation, resource reuse)
  - With resource sharing
- ➜ Intermediary notion of *partition* (defines ownership)
  - Each resource belongs to one, and only one, partition
  - Allocation = change of partitionning
  - No illusion of “resource creation”
  - Partition = unit to which resources are imputed/attributed
- Tasks can use *several* partitions (permission)
  - capability = right to use resources in a partition
  - e.g. right to write data, right to read&execute the code of shared libraries
  - right to change the sub-partitionning (for the allocation service)
- Definition: lending = transfer of permission, not of ownership
  - Dynamic use of resources
  - No intervention of the allocation module

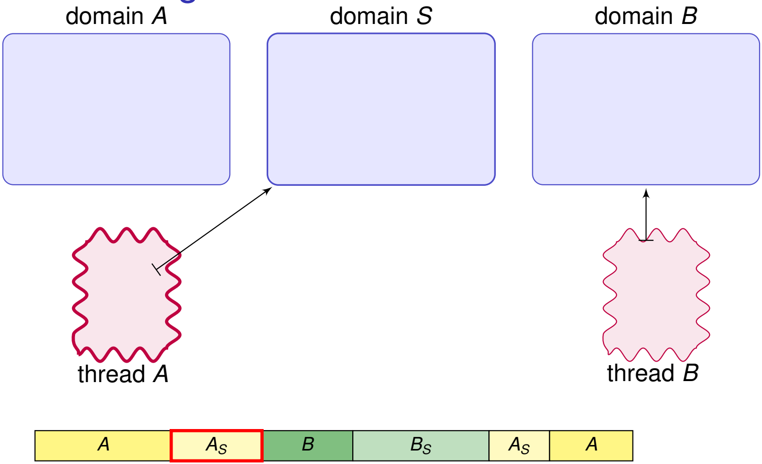
Thread lending: CPU time



- Thread = unit of CPU time dispatch

→ CPU time lending by thread lending

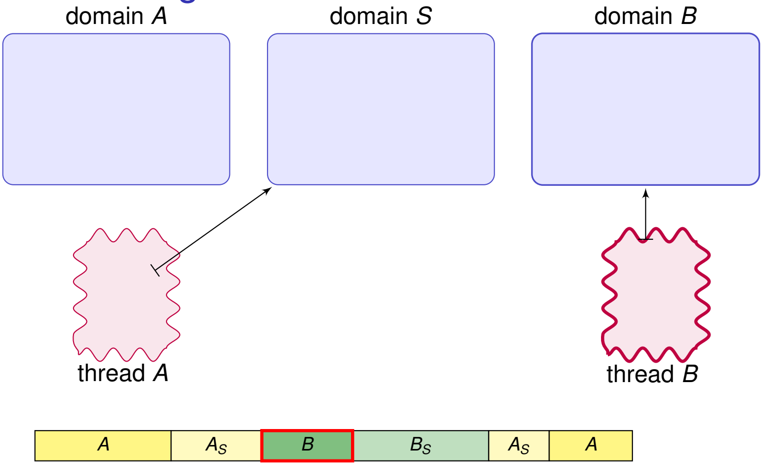
Thread lending: CPU time



- Thread = unit of CPU time dispatch

➜ CPU time lending by thread lending

Thread lending: CPU time

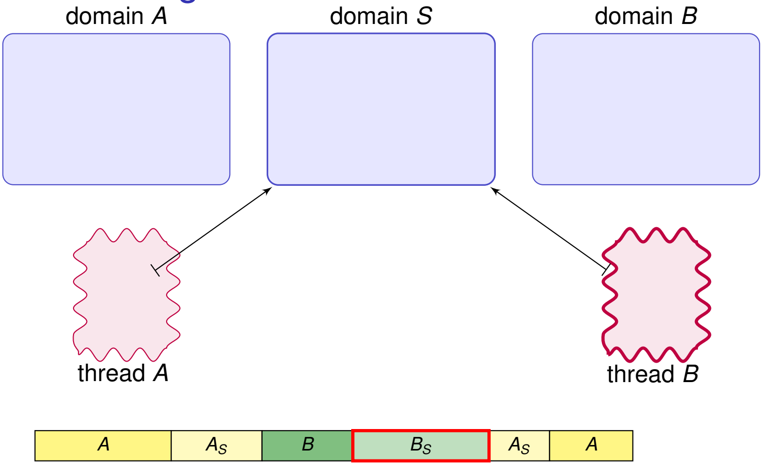


- Thread = unit of CPU time dispatch

➜ CPU time lending by thread lending

- Execution can stop and resume in the service
  - ➜ Multithreaded service

Thread lending: CPU time

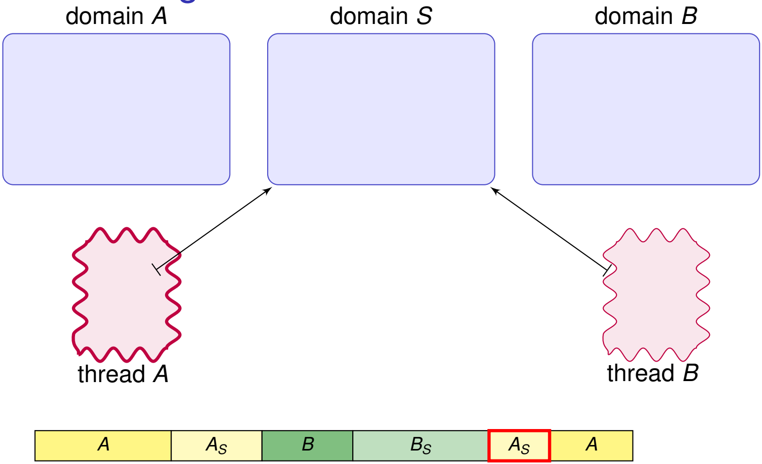


- Thread = unit of CPU time dispatch

➜ CPU time lending by thread lending

- Execution can stop and resume in the service
  - ➜ Multithreaded service

Thread lending: CPU time

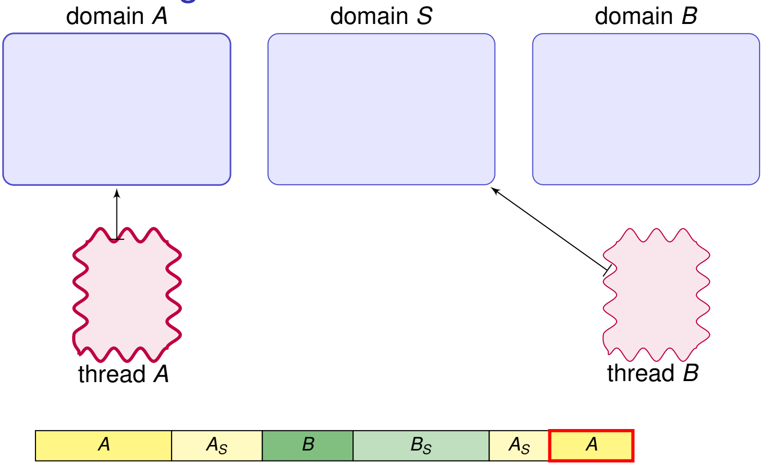


- Thread = unit of CPU time dispatch

➜ CPU time lending by thread lending

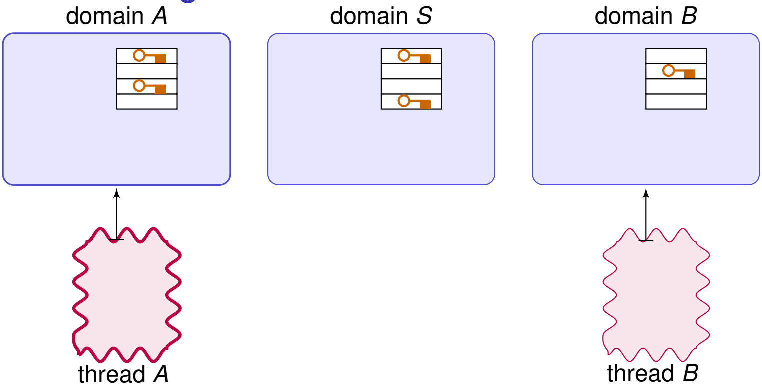
- Execution can stop and resume in the service
  - ➜ Multithreaded service

Thread lending: CPU time



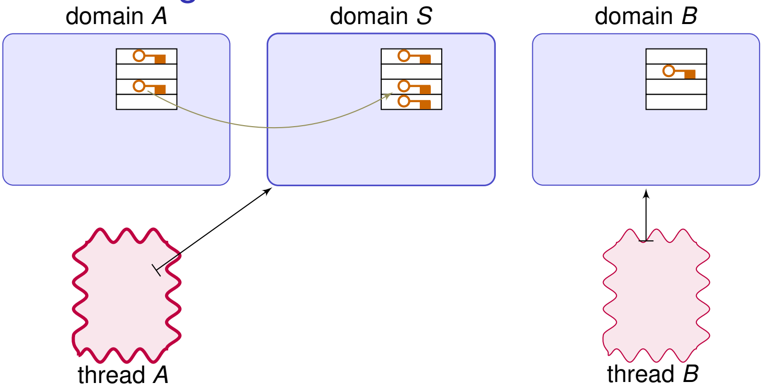
- Thread = unit of CPU time dispatch
- → CPU time lending by thread lending
- Execution can stop and resume in the service
  - → Multithreaded service

Thread lending: other resources



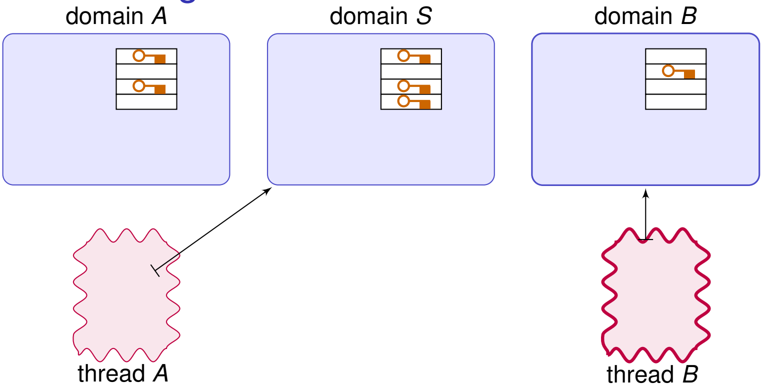
- Other resources must be lent (e.g. stack)
- Use a resource ⇒ own its key
- Usual approach: copy key to the service

Thread lending: other resources



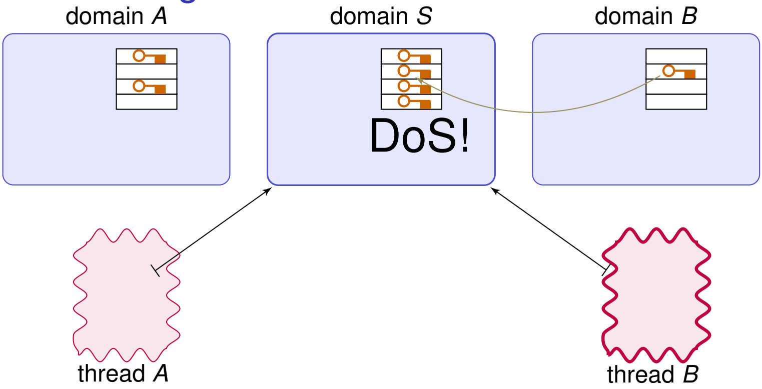
- Other resources must be lent (e.g. stack)
- Use a resource $\Rightarrow$ own its key
- Usual approach: copy key to the service

Thread lending: other resources



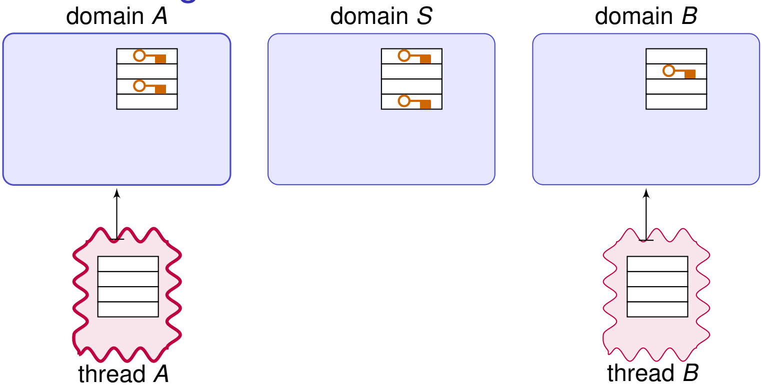
- Other resources must be lent (e.g. stack)
- Use a resource $\Rightarrow$ own its key
- Usual approach: copy key to the service

Thread lending: other resources



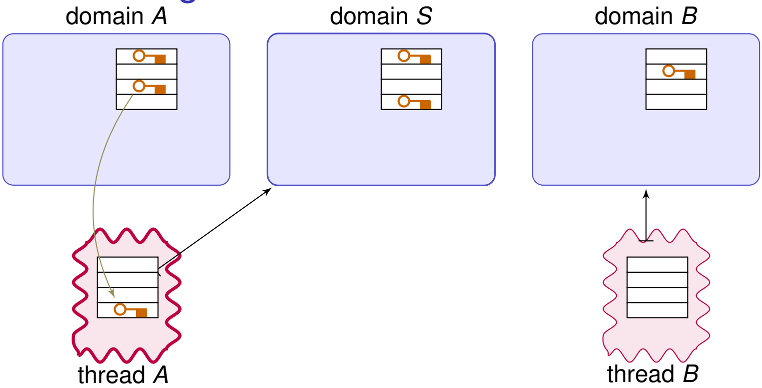
- Other resources must be lent (e.g. stack)
- Use a resource $\Rightarrow$ own its key
- Usual approach: copy key to the service
  - ➜ Storage by the service
  - ➜ DoS on the service memory
- Lending resource (to avoid DoS) can cause DoS!

Thread lending: other resources



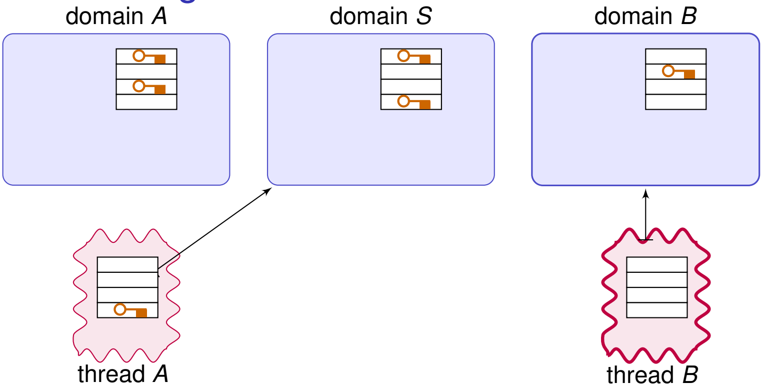
- Solution: also lend storage for keys (and other metadata)
  - ➜ Lent keys can be stored in per-thread storage
  - + Simple model (passive object call in OO)
  - Similar mechanism for memory mappings

Thread lending: other resources



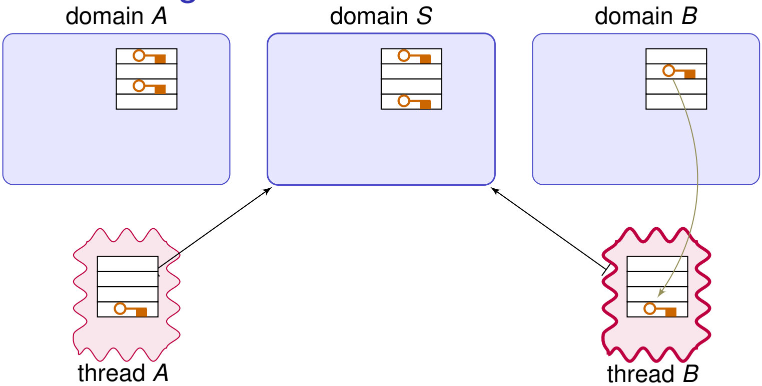
- Solution: also lend storage for keys (and other metadata)
  - ➜ Lent keys can be stored in per-thread storage
  - ✚ Simple model (passive object call in OO)
  - Similar mechanism for memory mappings

Thread lending: other resources



- Solution: also lend storage for keys (and other metadata)
  - ➜ Lent keys can be stored in per-thread storage
  - + Simple model (passive object call in OO)
  - Similar mechanism for memory mappings

Thread lending: other resources



- Solution: also lend storage for keys (and other metadata)
  - ➜ Lent keys can be stored in per-thread storage
  - + Simple model (passive object call in OO)
  - Similar mechanism for memory mappings
- Suppression de toute allocation pour la communication
  - , implemcapacitesno master object table
  - + No allocation by the service invulnerability to DoS

Plan

Proof of Programs

- Annotate source code by contracts, or spec's, with
  - Preconditions: what is supposed before the function call (`requires`)
  - Postconditions: what should be verified after the function call (`ensures`)
- Run automatic tool (like Frama-C / Jessie) which
  - Translates contracts into theorems, called proof obligations,
  - Proves them using automatic provers (like Alt-Ergo)
- Analyze proof failures (if any), complete specification
  - Loop invariants, assertions, etc.

Frama-C and ACSL language

- Frama-C : framework for analysis of C programs
  - Developed by CEA LIST and INRIA
  - Extensible plugin-oriented architecture
  - Open-source platform: http://frama-c.com
  - Includes various static and dynamic analyzers for C
    - Value analysis, test generation (PathCrawler), dependency, slicing...
- ACSL: ANSI/ISO C Specification Language
  - Common specification language for Frama-C analyzers
- Jessie plugin
  - Proof of programs (theorem proving)

Example : search in sorted array

```
//searches x in sorted array a of size l
int searchInArray(int* a, int l, int x){
  int k;

  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

a array of size l >= 0

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

a array of size l >= 0

a sorted

```c
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;      // not found
}
```

which (non local) variables can be modified?

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

First behavior:
If x present in a

then returned value
is index of x in a

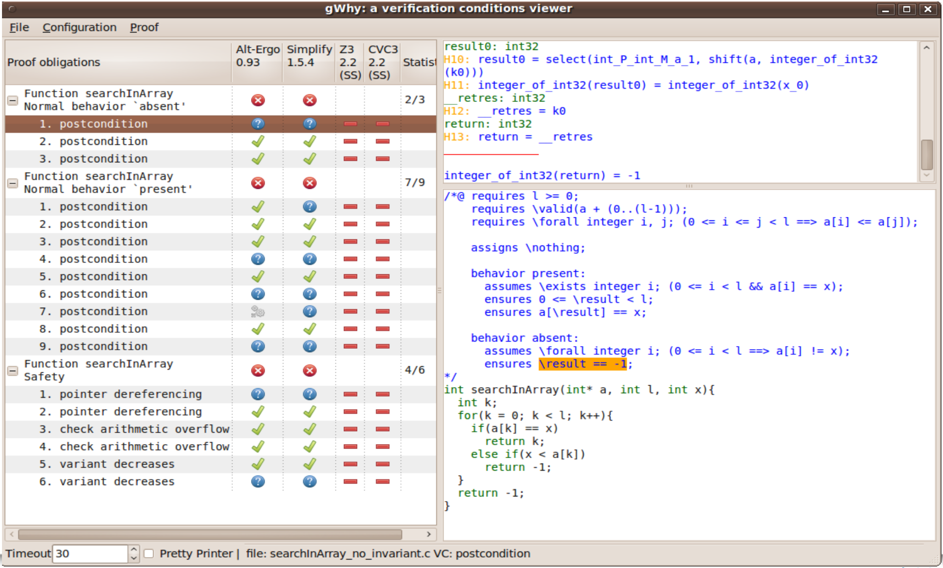
```c
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;      // not found
}
```

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

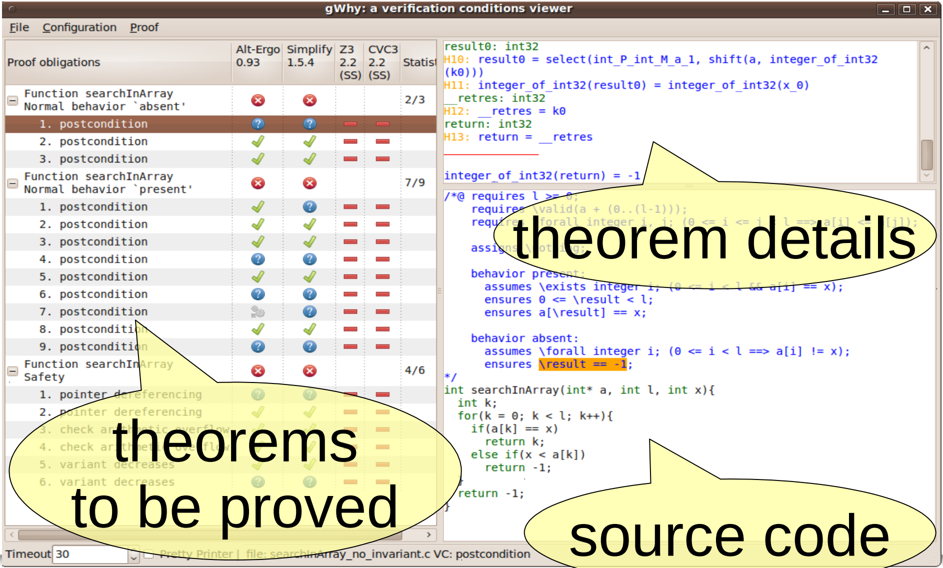
Second behavior:
If x absent in a

then returns -1

Jessie does not prove everything :



Jessie does not prove everything :



```c
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

```
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;   // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;      // not found
}
```

Difficulty: unknown number of loop iterations

Solution : Specify loop properties

```
/*@ ...
*/
int searchInArray(int* a, int l, int x){
  int k;
  /*@ loop invariant 0 <= k <= l &&
         \forall integer i; 0 <= i < k ==> a[i] < x;
      loop assigns \nothing;
      loop variant l-k;
  */
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

Solution : Specify loop properties

invariant: holds after k iterations

```
/*@ ...
*/
int searchInArray(int* a, int l, int x){
  int k;
  /*@ loop invariant 0 <= k <= l &&
          \forall integer i; 0 <= i < k ==> a[i] < x;
        loop assigns \nothing;
        loop variant l-k;
  */
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

Solution : Specify loop properties

invariant: holds after k iterations

does not assign variables

```
/*@ ...
*/
int searchInArray(int* a, int l, int x){
  int k;
  /*@ loop invariant 0 <= k <= l &&
         \forall integer i; 0 <= i < k ==> a[i] < x;
      loop assigns \nothing;
      loop variant l-k;
  */
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

Solution : Specify loop properties

```
/*@ ...
*/
int searchInArray(int* a, int l, int x){
  int k;
  /*@ loop invariant 0 <= k <= l &&
        \forall integer i; 0 <= i < k ==> a[i] < x;
      loop assigns \nothing;
      loop variant l-k;
  */
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;   // found, returns index
    else if(x < a[k])
      return -1 ;  // not found (a sorted)
  }
  return -1 ;      // not found
}
```

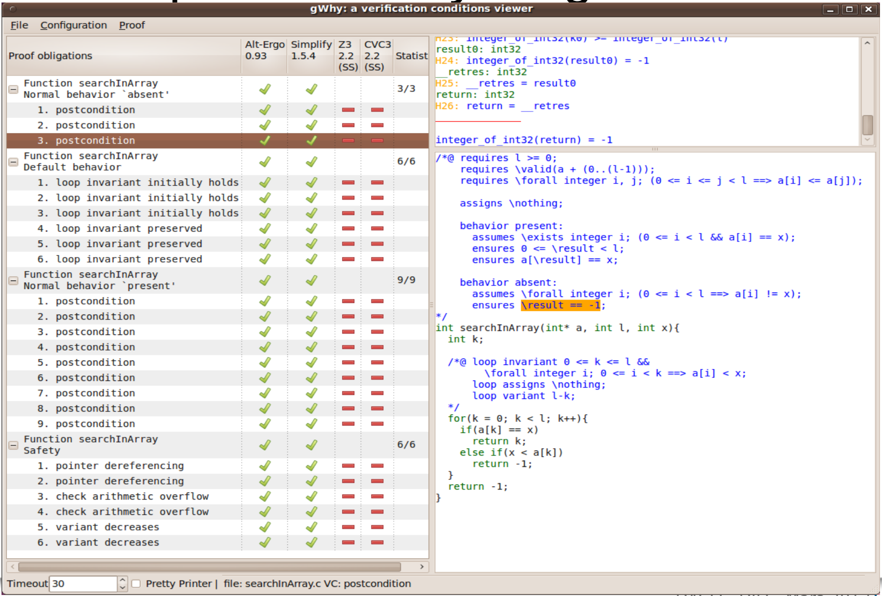
invariant: holds after k iterations

does not assign variables

variant: ≤ l−k more iterations

```c
/*@ requires l >= 0;
    requires \valid(a + (0..(l-1)));
    requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);

    assigns \nothing;

    behavior present:
      assumes \exists integer i; (0 <= i < l && a[i] == x);
      ensures 0 <= \result < l;
      ensures a[\result] == x;

    behavior absent:
      assumes \forall integer i; (0 <= i < l ==> a[i] != x);
      ensures \result == -1;
*/
int searchInArray(int* a, int l, int x){
  int k;
  /*@ loop invariant 0 <= k <= l &&
        \forall integer i; 0 <= i < k ==> a[i] < x;
      loop assigns \nothing;
      loop variant l-k;
  */
  for(k = 0; k < l; k++){
    if(a[k] == x)
      return k ;  // found, returns index
    else if(x < a[k])
      return -1 ; // not found (a sorted)
  }
  return -1 ;     // not found
}
```

Jessie proves everything now !



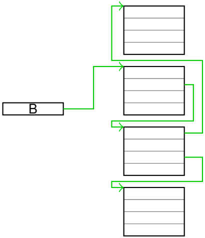
Plan

MMU: hardware mechanism for memory protection

- Splits memory into same-size *pages*
- Virtual memory roles:
  - Memory organization
  - Memory protection

➜ Hardware mechanism to restrict writing to a page: a page *p* is accessible iff:

1. The special register *B* points to a page *pd*,
2. That points to a page *pt*
3. That points to *p* (i.e. *p* is at level 3)



P_0 (accessible data)

P_1 (used as page directory)

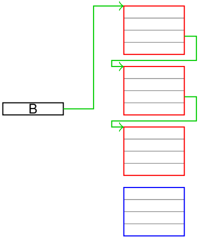
P_2 (used as page table)

P_3 (accessible data)

- Hypervisor must control what is written to page tables and directories

Hypervisor algorithm for memory isolation (1)

- Concept of *types* (PD,PT,D)
  - **Rule:** Pages may only be used according to their respective types
  - **Rule:** Pages of type PT and PD may only be changed by the hypervisor
- Dynamic usage of resources make attacks possible:
  - $\rightarrow$ change of type
  - A possible attack : a “data” page changes to type “PT”, then is used as a page table



P_0 (type PD, used as page directory)

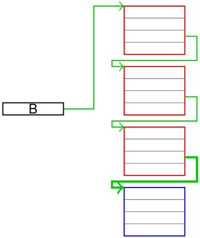
P_1 (type PT, used as page table)

P_2 (type D, accessible data)

P_3 (not owned, not accessible)

Hypervisor algorithm for memory isolation (1)

- Concept of *types* (PD,PT,D)
  - **Rule:** Pages may only be used according to their respective types
  - **Rule:** Pages of type PT and PD may only be changed by the hypervisor
- Dynamic usage of resources make attacks possible:
  - $\rightarrow$ change of type
  - A possible attack : a “data” page changes to type “PT”, then is used as a page table



P_0 (type PD, used as page directory)

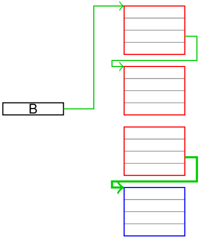
P_1 (type PT, used as page table)

P_2 (type D, accessible data)

P_3 (not owned, not accessible)

Hypervisor algorithm for memory isolation (1)

- Concept of *types* (PD,PT,D)
  - **Rule:** Pages may only be used according to their respective types
  - **Rule:** Pages of type PT and PD may only be changed by the hypervisor
- Dynamic usage of resources make attacks possible:
  - $\rightarrow$ change of type
  - A possible attack : a "data" page changes to type "PT", then is used as a page table



P_0 (type PD, used as page directory)

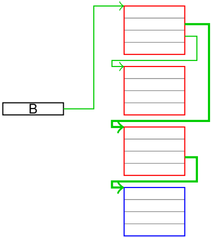
P_1 (type PT, used as page table)

P_2 (type PT, unused)

P_3 (not owned, not accessible)

Hypervisor algorithm for memory isolation (1)

- Concept of *types* (PD,PT,D)
  - **Rule:** Pages may only be used according to their respective types
  - **Rule:** Pages of type PT and PD may only be changed by the hypervisor
- Dynamic usage of resources make attacks possible:
  - $\rightarrow$ change of type
  - A possible attack : a “data” page changes to type “PT”, then is used as a page table



P_0 (type PD, used as page directory)

P_1 (type PT, used as page table)

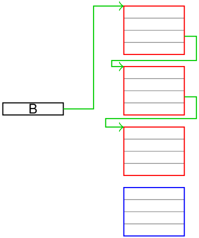
P_2 (type PT, used as page table)

P_3 (not owned, accessible data)

- Counter-measure: changing type requires “cleanup”

Hypervisor algorithm for memory isolation (2)

- Insufficient counter-measure
- Other attack: pages used both as “data” (accessibles) and “pagetable”.



P_0 (type PD, used as page directory)

P_1 (type PT, used as page table)

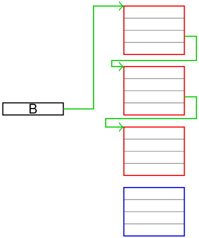
P_2 (type D, accessible data)

P_3 (not owned, not accessible)

- Possible attack: a page used as “data”, change type to “PT” (with cleanup), used as pagetable, then directly changed

Hypervisor algorithm for memory isolation (2)

- Insufficient counter-measure
- Other attack: pages used both as “data” (accessibles) and “pagetable”.



P_0 (type PD, used as page directory)

P_1 (type PT, used as page table)

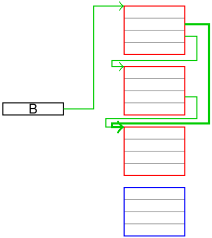
P_2 (type PT, accessible data)

P_3 (not owned, not accessible)

  - Possible attack: a page used as “data”, change type to “PT” (with cleanup), used as pagetable, then directly changed

Hypervisor algorithm for memory isolation (2)

- Insufficient counter-measure
- Other attack: pages used both as “data” (accessibles) and “pagetable”.



P_0 (type PD, used as page directory)

P_1 (type PT, used as page table)

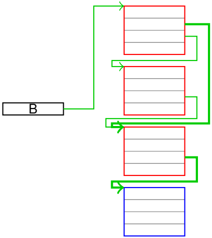
P_2 (type PT, accessible data, used as page table)

P_3 (not owned, not accessible)

- Possible attack: a page used as “data”, change type to “PT” (with cleanup), used as pagetable, then directly changed

Hypervisor algorithm for memory isolation (2)

- Insufficient counter-measure
- Other attack: pages used both as “data” (accessibles) and “pagetable”.



P_0 (type PD, used as page directory)

P_1 (type PT, used as page table)

P_2 (type PT, accessible data, used as page table)

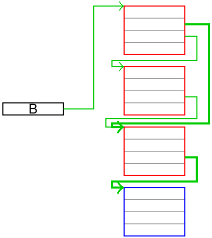
P_3 (not owned, accessible data)

- Possible attack: a page used as “data”, change type to “PT” (with cleanup), used as pagetable, then directly changed
- Counter-measure:
  - Account for number of mappings to a page
  - Allow changing types only if number of mappings = 0

- Is it still possible to break the **Rules**?

Hypervisor algorithm for memory isolation (2)

- Insufficient counter-measure
- Other attack: pages used both as “data” (accessibles) and “pagetable”.



P_0 (type PD, used as page directory)

P_1 (type PT, used as page table)

P_2 (type PT, accessible data, used as page table)

P_3 (not owned, accessible data)

  - Possible attack: a page used as “data”, change type to “PT” (with cleanup), used as pagetable, then directly changed
  - Counter-measure:
    - Account for number of mappings to a page
    - Allow changing types only if number of mappings = 0

- Is it still possible to break the **Rules**?

➜ No (formally proved)

Verification of Virtual Memory Module

- We specify a module prototype and prove it in Frama-C / Jessie

What to do with proof failures ?

- Proof failures come from complex inductive predicates
- They can be proved interactively in Coq (long, expensive)
- Or...

```
/*@ inductive MappingsAllOverOnePage{L}(integer pageIndex, integer lastIndex, integer refferredIndex, integer mappingsNum){
  @   case oneEq:
  @     \forall integer pageIndex, refferredIndex;
  @       0<=pageIndex<NumOfPages && 0<=refferredIndex<NumOfPages && pContents[pageIndex*PageSizeWords] == refferredIndex ==>
  @       MappingsAllOverOnePage(pageIndex, 0, refferredIndex, 1);

  @   case oneNotEq:
  @     \forall integer pageIndex, refferredIndex;
  @       0<=pageIndex<NumOfPages && 0<=refferredIndex<NumOfPages && pContents[pageIndex*PageSizeWords] != refferredIndex  ==>
  @       MappingsAllOverOnePage(pageIndex, 0, refferredIndex, 0);

  @   case severalLastNotEq:
  @     \forall integer pageIndex, lastIndex, refferredIndex, mappingsNum;
  @       (0<=pageIndex<NumOfPages && 0<=refferredIndex<NumOfPages && 0 < lastIndex < PageSizeWords &&
  @       mappingsNum >=0 && pContents[pageIndex*PageSizeWords + lastIndex] != refferredIndex &&
  @       MappingsAllOverOnePage(pageIndex, lastIndex-1, refferredIndex, mappingsNum) ==>
  @       MappingsAllOverOnePage(pageIndex, lastIndex, refferredIndex, mappingsNum) );

  @   case severalLastEq:
  @     \forall integer pageIndex, lastIndex, refferredIndex, mappingsNum;
  @       (0<=pageIndex<NumOfPages && 0<=refferredIndex<NumOfPages && 0 < lastIndex < PageSizeWords &&
  @       mappingsNum >=0 && pContents[pageIndex*PageSizeWords + lastIndex] == refferredIndex &&
  @       MappingsAllOverOnePage(pageIndex, lastIndex-1, refferredIndex, mappingsNum) ==>
  @       MappingsAllOverOnePage(pageIndex, lastIndex, refferredIndex, mappingsNum+1) );
  @ }
  @
  @ inductive MappingsAllOverAllPages{L}(integer lastPage, integer refferredIndex, integer mappingsNum){
  @   case onePage:
  @     \forall integer refferredIndex, mappingsNum;
          ( 0<=refferredIndex<NumOfPages &&
  @       MappingsAllOverOnePage(0, PageSizeWords-1, refferredIndex, mappingsNum) ) ==>
  @       MappingsAllOverAllPages(0, refferredIndex, mappingsNum);
  @   case severalPages:
  @     \forall integer lastPage, refferredIndex, mappingsNumPrevPages, mappingsNumLastPage;
  @       ( 0<=refferredIndex<NumOfPages && 0 < lastPage < NumOfPages ==>
  @       MappingsAllOverAllPages(lastPage-1, refferredIndex, mappingsNumPrevPages) &&
  @       MappingsAllOverOnePage(lastPage, PageSizeWords-1, refferredIndex, mappingsNumLastPage) ==>
  @       MappingsAllOverAllPages(lastPage, refferredIndex, mappingsNumPrevPages+mappingsNumLastPage) );
  @ }
*/
```

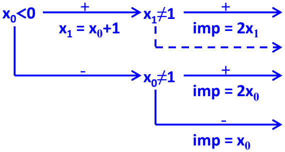
Testing to complete the proof

- Isolate unproved parts in the smallest possible functions
- Write (automatically generate with E-ACSL) C specification : pre/post
- Use cross-checking to verify conformity
  - Exhaustive path exploration, and even more :
  - (Function paths) X (Spec paths)
- If necessary, reduce search space
  - Consider a smaller number of pages
  - Consider a smaller page size

PathCrawler testing tool

- Concolic /DSE testing tool for C developed at CEA LIST
- Input: a complete compilable source code
- Automatically creates test cases to cover program paths
- Uses code instrumentation, concrete and symbolic execution, constraint solving
- Exact semantics: don't rely on concrete values to approximate the path predicate
- Similar to PEX, DART/CUTE, KLEE, SAGE etc.

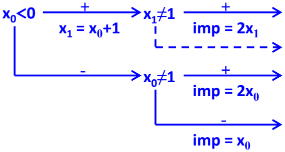
Cross-checking conformity with a specification



implementation

```
int f(int x){
 if(x < 0)
  x = x + 1;
 if(x != 1)
  x = 2*x;
 return x; }
```

Cross-checking conformity with a specification



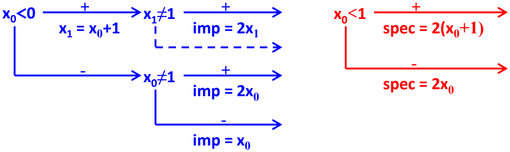
implementation

```
int f(int x){
 if(x < 0)
  x = x + 1;
 if(x != 1)
  x = 2*x;
 return x; }
```

specification

If x is less than 1 then
the result should be 2(x + 1)
else the result should be 2x

Cross-checking conformity with a specification



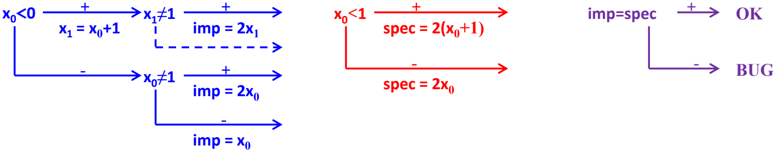
implementation

```
int f(int x){
 if(x < 0)
  x = x + 1;
 if(x != 1)
  x = 2*x;
 return x; }
```

specification

```
int spec_f(int x){
 if(x < 1)
  x = 2*(x + 1);
 else
  x = 2*x;
 return x; }
```

Cross-checking conformity with a specification



implementation

```
int f(int x){
 if(x < 0)
  x = x + 1;
 if(x != 1)
  x = 2*x;
 return x; }
```

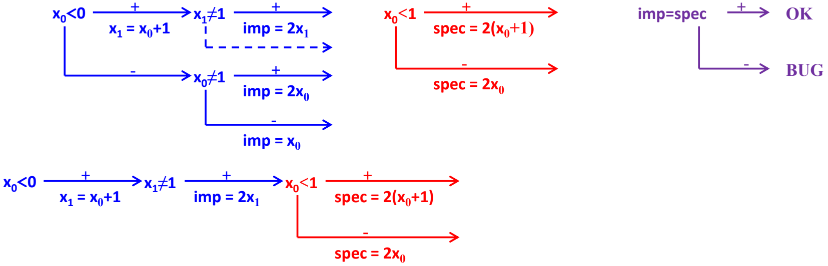
specification

```
int spec_f(int x){
 if(x < 1)
  x = 2*(x + 1);
 else
  x = 2*x;
 return x; }
```

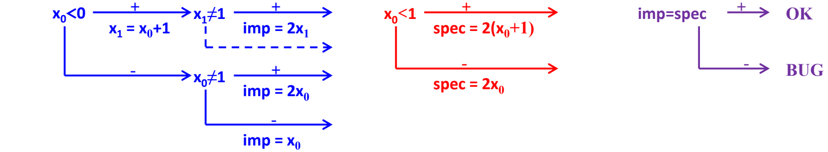
comparison

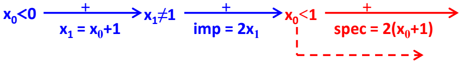
```
int cross_f(int x){
 int imp = f(x);
 int spec=spec_f(x);
 if(imp!=spec)
   return 0;
 else return 1; }
```

Cross-checking conformity with a specification



Cross-checking conformity with a specification

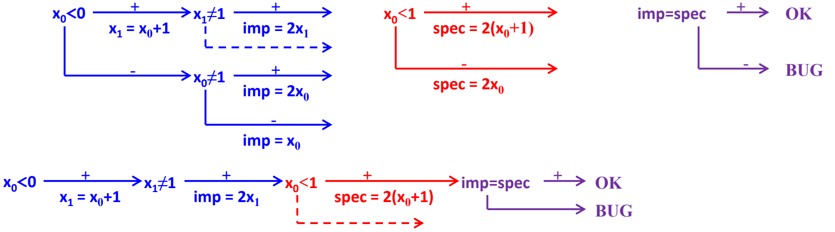




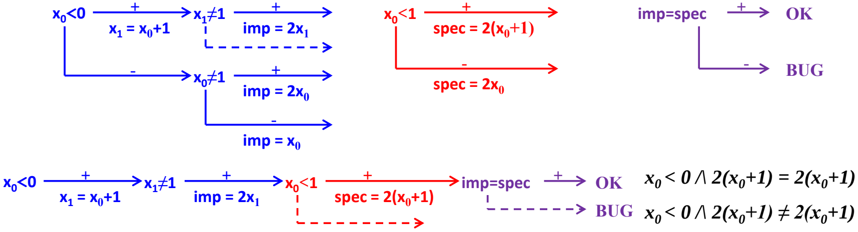
$$x_0 < 0 \wedge (x_0 + 1) \neq 1 \wedge x_0 < 1 \rightarrow x_0 < 0$$

$$x_0 < 0 \wedge (x_0 + 1) \neq 1 \wedge x_0 \geq 1$$

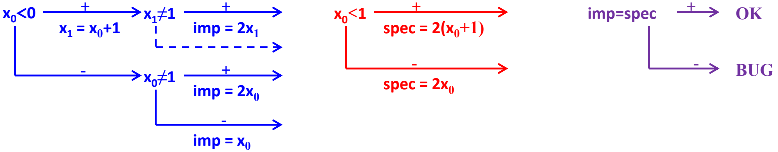
Cross-checking conformity with a specification

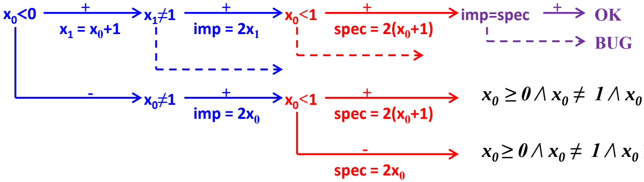


Cross-checking conformity with a specification



Cross-checking conformity with a specification



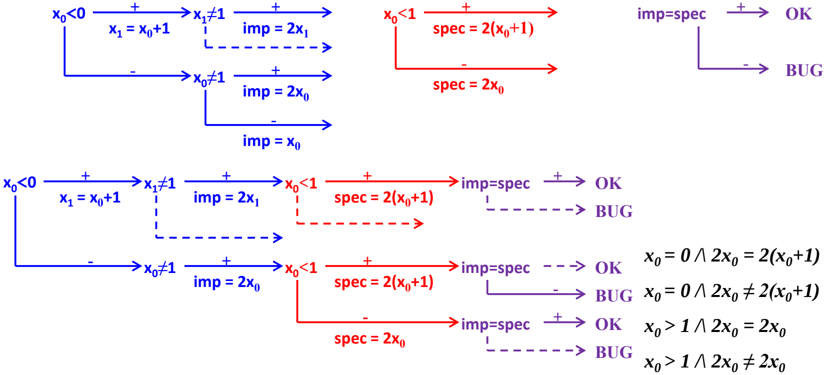


$x_0 < 0$

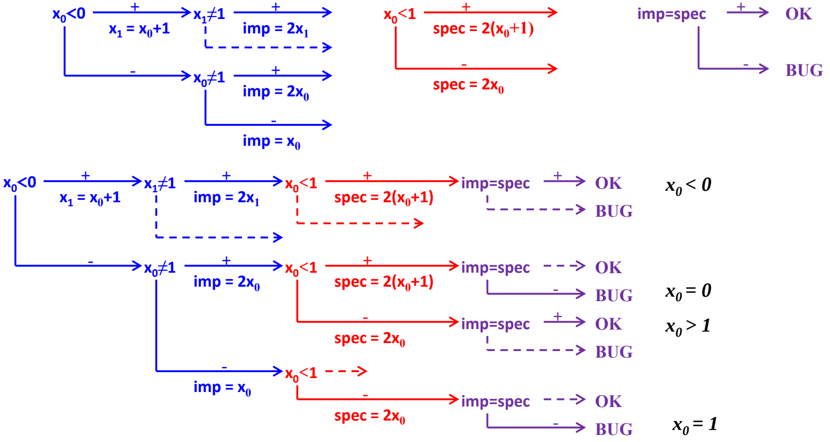
$x_0 \geq 0 \wedge x_0 \neq 1 \wedge x_0 < 1 \rightarrow x_0 = 0$

$x_0 \geq 0 \wedge x_0 \neq 1 \wedge x_0 \geq 1 \rightarrow x_0 > 1$

Cross-checking conformity with a specification



Cross-checking conformity with a specification



Proving the VM module with Frama-C: results

- Prove that all functions fulfill their specifications
- Prove that the **Rules** hold $\rightarrow$ proof of memory isolation
- Statistics:
  - 2000 LOC, 80% spec, 20% C code
  - 37 functions, 3969 properties to be proved
  - 3915 properties (98.8%) proved with Jessie
- Proof-of-concept, much work remains:
  - Modeling hardware mechanisms (e.g. TLB cache)
  - Proof of multicore version (uses lock-free algorithms)
  - Parts of the code have two versions:
    - "simple" (automatically provable)
    - "fast" (efficient, but requires more proof effort)
  - Proof of remaining 1.2% using interactive theorem prover

Plan

1. Anaxagoros: a secure hypervisor for the cloud
2. Proof of programs with Frama-C
3. Verification of a hypervisor algorithm
4. **Conclusion**

Summary

- Anaxagoros: a secure foundation for the Cloud
  - Provides maximum isolation between tasks or VMs
  - Strong focus on resource security
  - Allows reusability/ease of use through virtualization
  - Minimizes the amount of trusted code $\rightarrow$
    - minimize bugs and security breaches
    - amenable to formal verification
- Formal verification technology is becoming applicable
  - Formal proof provides the *highest* level of confidence in a program
  - Tools such as Frama-C are now able to prove actual algorithms with feasible effort
  - Requires an important effort; reasonable only if hypervisor is designed to be proved (size, cleanness of internal interfaces)
  - Other verification techniques in Frama-C applicable with less effort (test generation, abstract interpretation...)

Perspectives

- Continue improving Anaxagoros:
  - Improve performance, in particular on specific industrial cases
  - Study hardware breaches in performance isolation (e.g. cache partitioning, limitation of preemptions)
- Continue the proof effort
  - Use interactive proof assistant (Coq) for the 1.2% unproved theorems
  - On-going research efforts:
    - Proving parallel algorithms
    - Maintainability: updating the proof when the code changes
- Industrial offer with a CEA startup being created around the Frama-C technology
  - Help industry to use formal methods for cybersecurity

leti

LABORATOIRE D'ÉLECTRONIQUE
ET DE TECHNOLOGIES
DE L'INFORMATION

list

LABORATOIRE D'INTÉGRATION
DES SYSTÈMES
ET DES TECHNOLOGIES

Thank you

INSTITUT CARNOT CEA LETI

MINATEC

cea

INSTITUT CARNOT CEA LIST

digiteo

LLOYD'S REGISTER QUALITY ASSURANCE · ISO9001