A Declarative Language Approach to Device Configuration

Adrian Schüpbach Andrew Baumann Timothy Roscoe Simon Peter
Systems Group, ETH Zurich

This talk is about

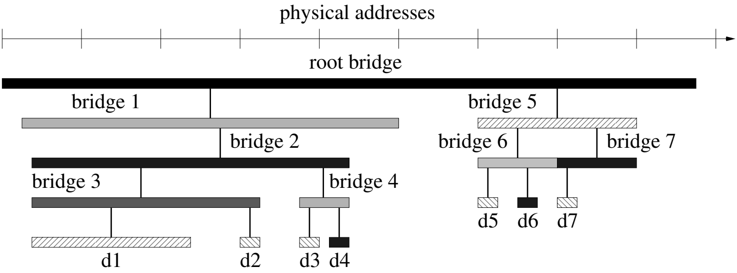
- Hardware resource configuration is harder than you think
  1. The idealized problem is complex
  2. In practice there are many exceptions and quirks
- We apply high-level languages to deal with hardware configuration
  - Approach
  - Evaluation

Hardware configuration is surprisingly complex!

- Allocate hardware resources to devices
  - Physical address ranges
  - RAM buffers
  - Interrupt lines
  - ...
- These resources are limited
- The problem is constrained in multiple ways
- Hardware in reality does not fit the specifications, it often has bugs

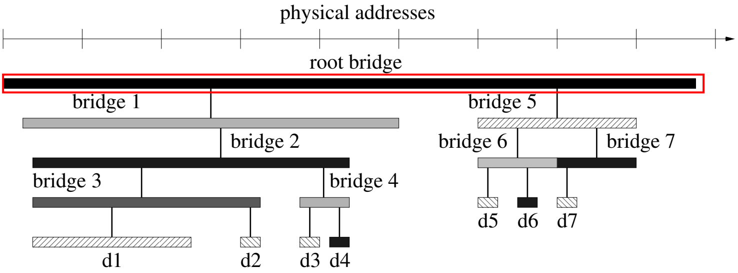
Example: PCI bus configuration

- Tree with multiple children per node
  - Inner nodes: PCI bridges
  - Leaves: devices
  - PCI bridge hierarchy translates physical addresses on device requests
  - Base address registers (BARs) define base address



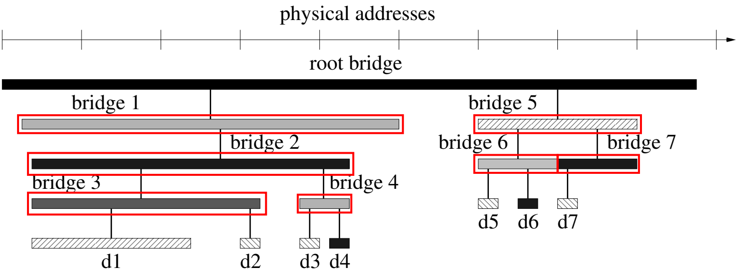
Example: PCI bus configuration

- Tree with multiple children per node
  - Inner nodes: PCI bridges
  - Leaves: devices
  - PCI bridge hierarchy translates physical addresses on device requests
  - Base address registers (BARs) define base address



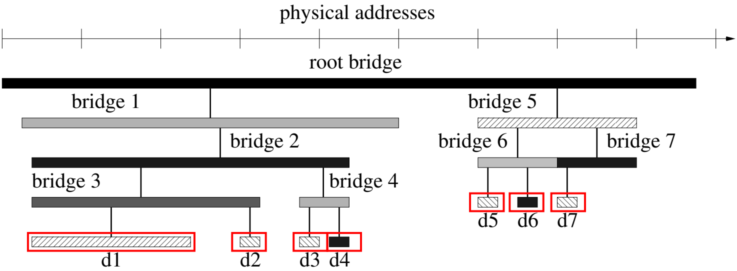
Example: PCI bus configuration

- Tree with multiple children per node
  - **Inner nodes: PCI bridges**
  - Leaves: devices
  - PCI bridge hierarchy translates physical addresses on device requests
  - Base address registers (BARs) define base address



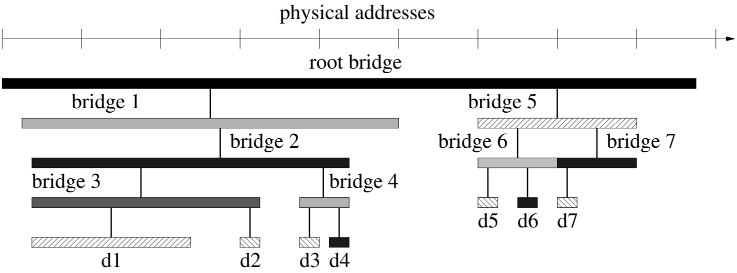
Example: PCI bus configuration

- Tree with multiple children per node
  - Inner nodes: PCI bridges
  - **Leaves: devices**
  - PCI bridge hierarchy translates physical addresses on device requests
  - Base address registers (BARs) define base address



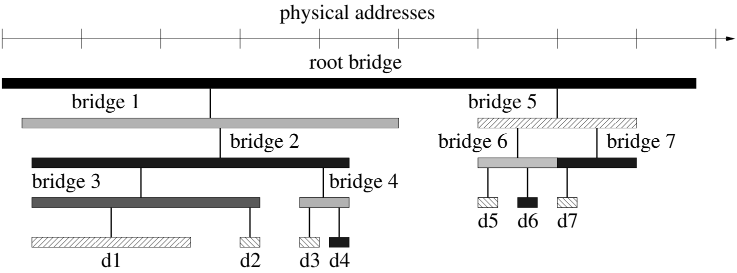
Example: PCI bus configuration

- Tree with multiple children per node
  - Inner nodes: PCI bridges
  - Leaves: devices
  - **PCI bridge hierarchy translates physical addresses on device requests**
  - Base address registers (BARs) define base address



Example: PCI bus configuration

- Tree with multiple children per node
  - Inner nodes: PCI bridges
  - Leaves: devices
  - PCI bridge hierarchy translates physical addresses on device requests
  - **Base address registers (BARs) define base address**

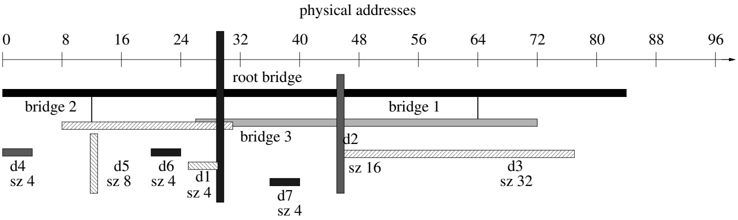


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

1. Uninitialized PCI bus

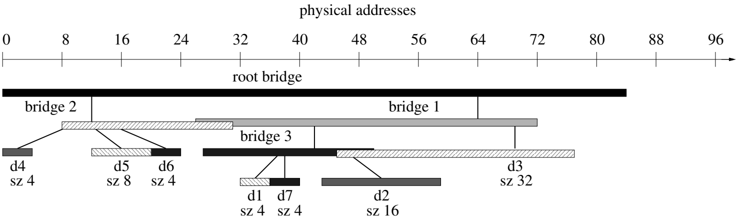


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

2. All devices should be configured

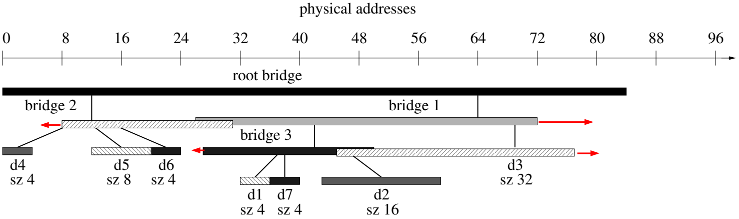


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

3. No overlapping of siblings must occur

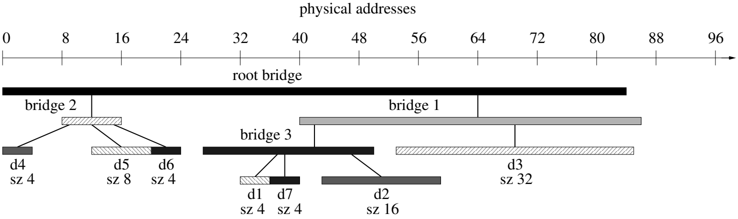


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

3. No overlapping of siblings must occur

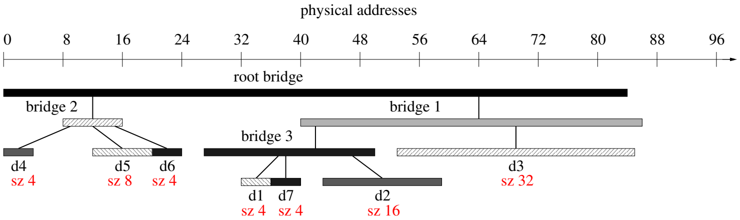


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

4. Device addresses have to be naturally aligned

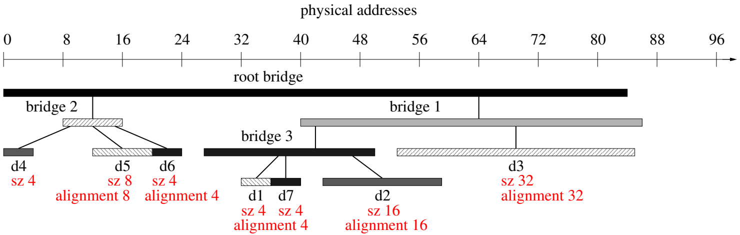


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

4. Device addresses have to be naturally aligned

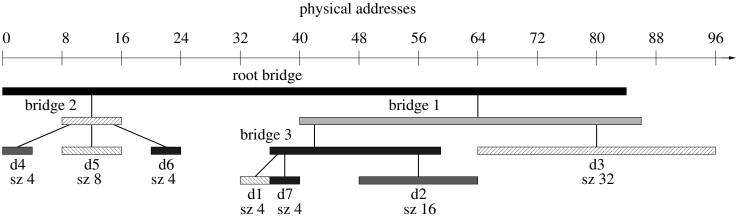


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

4. Device addresses have to be naturally aligned

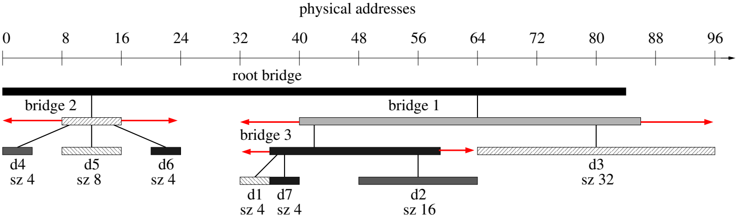


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

5. Children have to be within their parent bridges address range

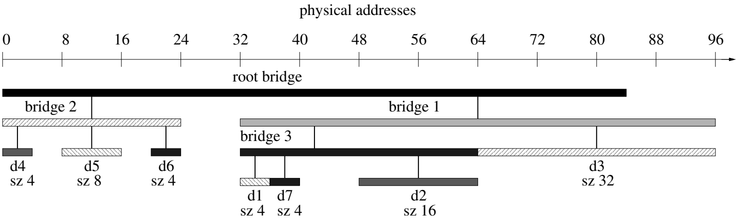


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

5. Children have to be within their parent bridges address range

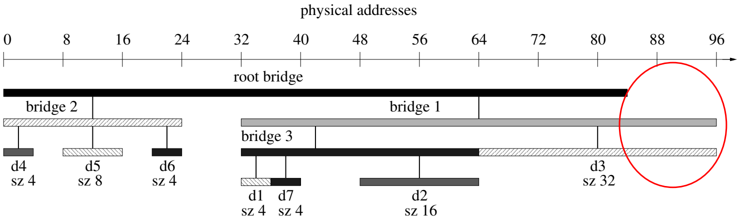


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

6. The PCI tree has to fit within available address range

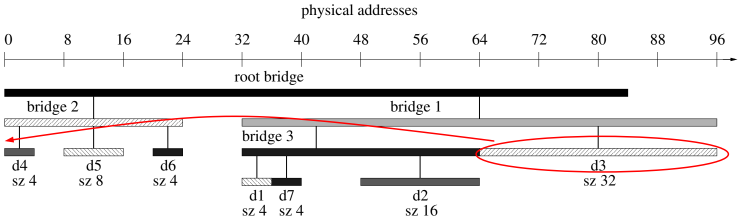


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

6. The PCI tree has to fit within available address range

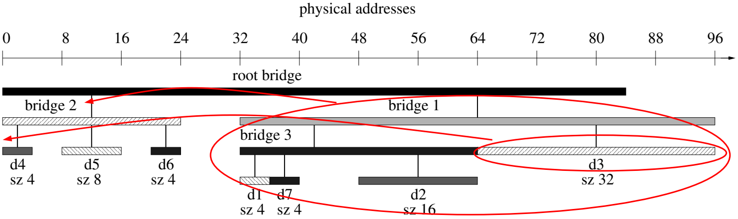


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

6. The PCI tree has to fit within available address range

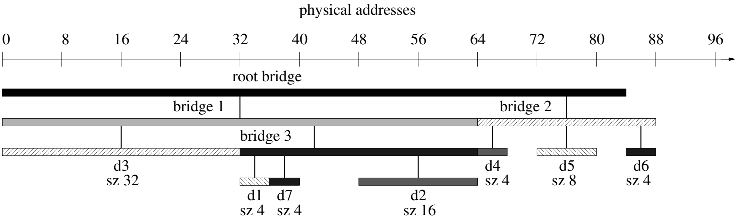


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

6. The PCI tree has to fit within available address range

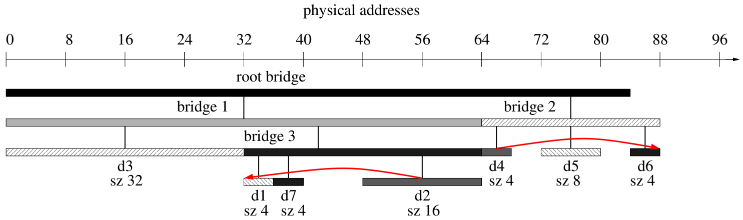


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules.

6. The PCI tree has to fit within available address range

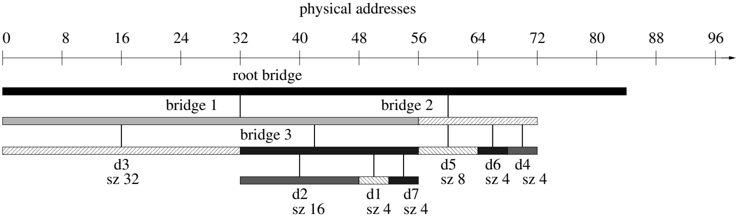


The Problem

Hardware resource allocation in PCI

In theory, apply the following rules. Until here idealized problem.

6. The PCI tree has to fit within available address range

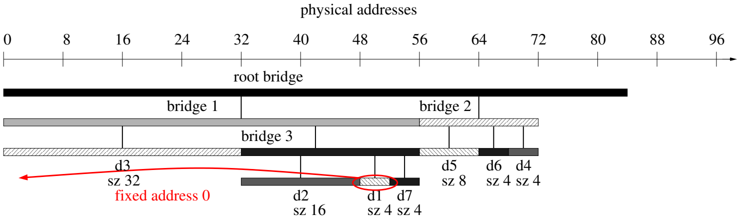


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

7. Some devices have fixed address requirements

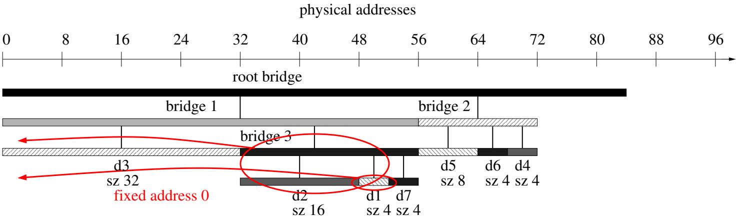


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

7. Some devices have fixed address requirements

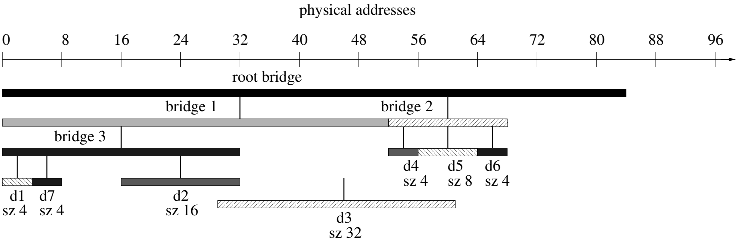


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

7. Some devices have fixed address requirements

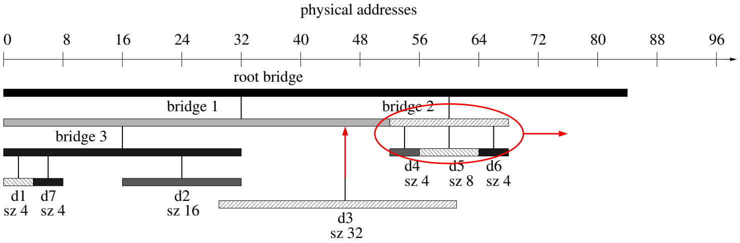


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

7. Some devices have fixed address requirements

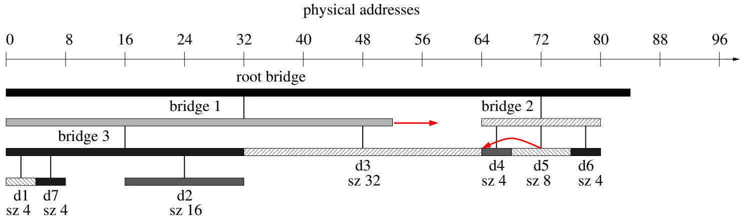


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

7. Some devices have fixed address requirements

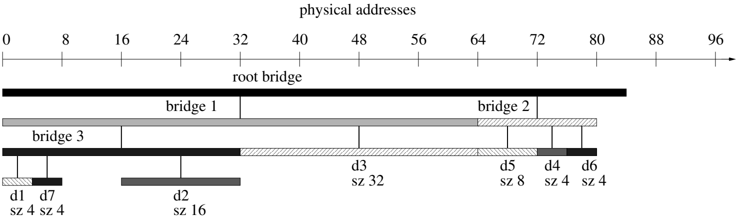


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

7. Some devices have fixed address requirements

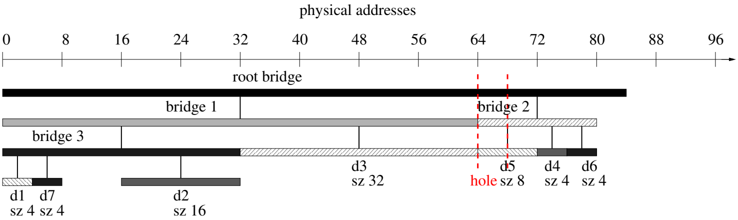


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

8. Physical memory regions have holes

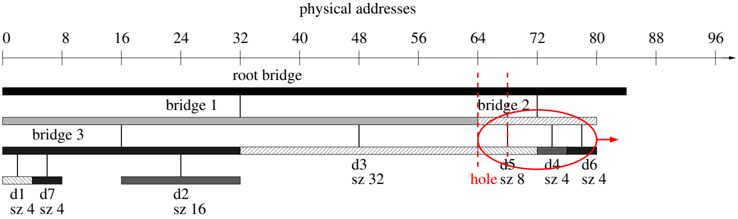


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

8. Physical memory regions have holes

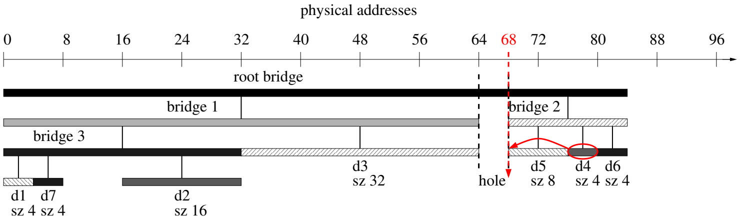


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

8. Physical memory regions have holes

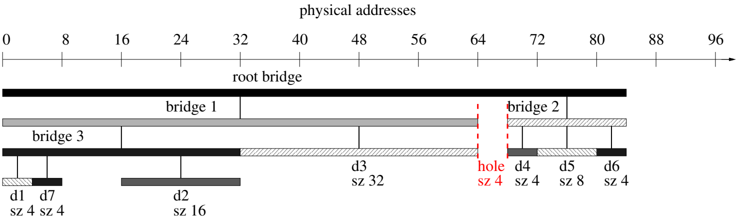


The Problem

Hardware resource allocation in PCI

But in practice handle also special cases.

8. Physical memory regions have holes



Quirks (some of the 3000 LOCs in Linux's quirks.c)

-
  ```
  /*Following the PCI ordering rules is optional on the
  AMD762.  I'm not sure what the designers were smoking
  but let's not inhale...
  To be fair to AMD, it follows the spec by default, its
  BIOS people who turn it off!  */
  ```

Quirks (some of the 3000 LOCs in Linux's quirks.c)

- /*Following the PCI ordering rules is optional on the AMD762. I'm not sure what the designers were smoking but let's not inhale...

  To be fair to AMD, it follows the spec by default, its BIOS people who turn it off! */
- This card decodes and responds to addresses not apparently assigned to it. We force a larger allocation to ensure that nothing gets put too close to it.

Quirks (some of the 3000 LOCs in Linux's quirks.c)

- /*Following the PCI ordering rules is optional on the AMD762. I'm not sure what the designers were smoking but let's not inhale...

  To be fair to AMD, it follows the spec by default, its BIOS people who turn it off! */
- This card decodes and responds to addresses not apparently assigned to it. We force a larger allocation to ensure that nothing gets put too close to it.
- S3 868 and 968 chips report region size equal to 32M, but they decode 64M.

Quirks (some of the 3000 LOCs in Linux's quirks.c)

- /*Following the PCI ordering rules is optional on the AMD762. I'm not sure what the designers were smoking but let's not inhale...

  To be fair to AMD, it follows the spec by default, its BIOS people who turn it off! */
- This card decodes and responds to addresses not apparently assigned to it. We force a larger allocation to ensure that nothing gets put too close to it.
- S3 868 and 968 chips report region size equal to 32M, but they decode 64M.
- the first BAR is the location of the IO APIC...we must not touch this (and it's already covered by the fixmap), so forcibly insert it into the resource tree, The next five BARs all seem to be rubbish, so just clean them out

Quirks (some of the 3000 LOCs in Linux's quirks.c)

- /*Following the PCI ordering rules is optional on the AMD762. I'm not sure what the designers were smoking but let's not inhale...

  To be fair to AMD, it follows the spec by default, its BIOS people who turn it off! */
- This card decodes and responds to addresses not apparently assigned to it. We force a larger allocation to ensure that nothing gets put too close to it.
- S3 868 and 968 chips report region size equal to 32M, but they decode 64M.
- **the first BAR is the location of the IO APIC...we must not touch this (and it's already covered by the fixmap), so forcibly insert it into the resource tree, The next five BARs all seem to be rubbish, so just clean them out**

What do people do today?

- Linux uses BIOS allocation and runs fixup procedure
  - Configure missing devices
  - Allocate address range from bridge, or fail if bridge does not have enough free address range
- Windows Vista, Server 2008: PCI Multi-Level Rebalance
  - Can move bridges to a place with bigger free space
- IBM US patent 5,778,197, 1998: Method for allocating system resources in a hierarchical bus structure
  - Recursive bottom-up algorithm to allocate resources

What we wanted to try

- Express allocation problem as constraint logic program (CLP) in high-level language
- Explore modern techniques to configure hardware
- Separate allocation computation from register access

- Why CLP?
  - Allows constraining variables before assigning concrete values
  - Natural way to implement allocation rules
  - Naturally express hardware constraints and limitations
  - Handle quirks in a clean way, not ad-hoc
  - Leads to platform independence and portability

- We use ECL^{i}PS^{e}: Prolog + constraint extensions

How does CLP work?

1. Create tree data structure which matches the PCI tree
2. Create Base and Size variables in every node in the data structure
3. Apply constraints to these variables
4. Instantiate the variables with concrete values representing PCI base adresses

Allocation rule: siblings must not overlap

Code written in ECL^{i}PS^e

```
nonoverlap(Tree) :-
  % collect direct children of this root in ChildList
  t(_ ,Children) = Tree,
  maplist(root,Children,ChildList),
  % if there are direct children...
  ( not ChildList=[] ->
      % determine base and size of each child
      maplist(base,ChildList,Bases),
      maplist(size,ChildList,Sizes),
      % constrain the regions they define not to overlap
      disjunctive(Bases,Sizes)
    ; true
  ),
  % recurse on all children
  ( foreach(El, Children) do nonoverlap(El) ).
```

Quirk: do not move BARs pointing to IOAPICs

Code written in ECL^{i}PS^e

```
keep_ioapic_bars(_, []).
keep_ioapic_bars(Buselements, [H|IOAPICList]) :-
  ( % get the base of the first IOAPIC
    range(B, _) = H,
    % check if a BAR with the same original base exists
    bar(addr(Bus,Dev,Fun),_,OrigBase,_,_,_,_),
    OrigBase =:= B ->
    % if found, keep the device at its original address
    keep_orig_addr(Buselements, _, _, _, Bus, Dev, Fun);
    true
  ),
  % iterate on the IOAPIC list
  keep_ioapic_bars(Buselements, IOAPICList).
```

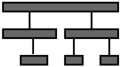
Implementation

- We program the PCI bus in our research operating system Barrelfish like this
- We use ECL^{i}PS^{e}-CLP engine to run the algorithm
  - Starts early in the operating system boot sequence
  - Uses a RAM disk to load everything necessary
  - Is self-contained

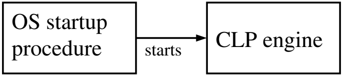
Boot sequence

OS startup procedure

PCI bus



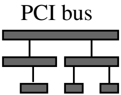
Boot sequence



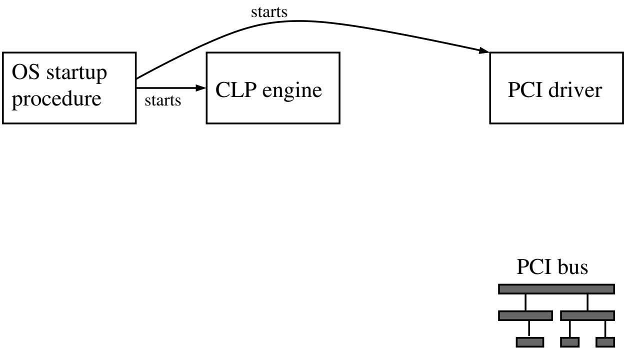
OS startup procedure

starts

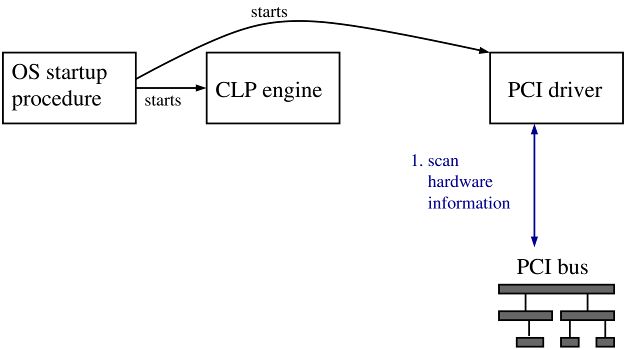
CLP engine



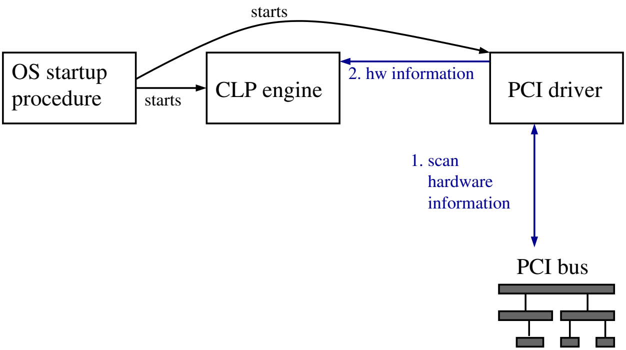
Boot sequence



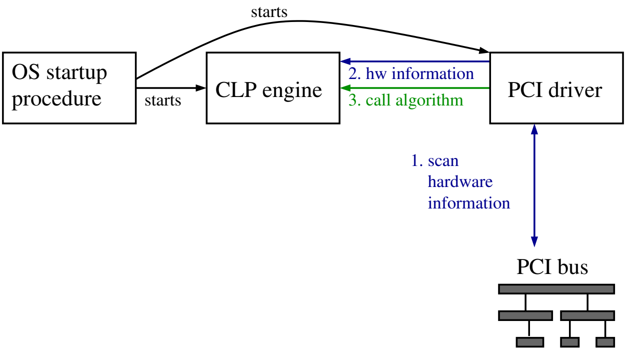
Boot sequence



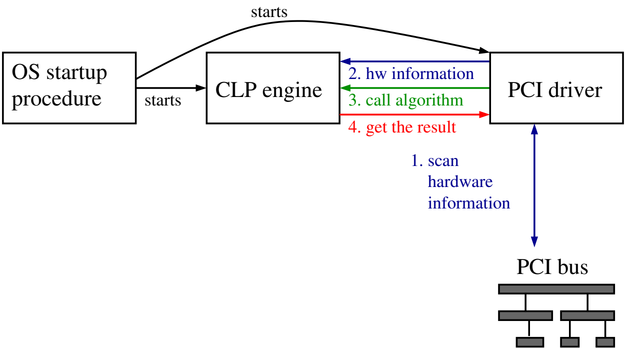
Boot sequence



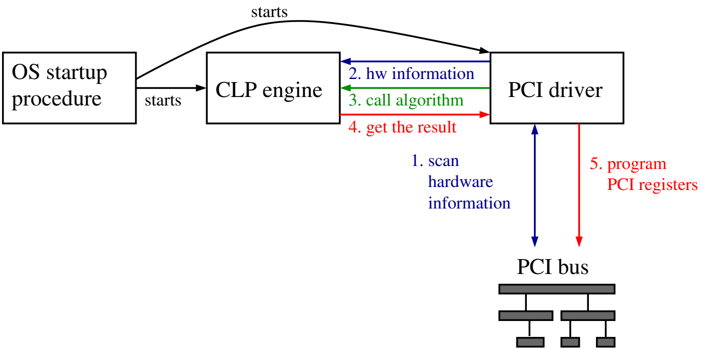
Boot sequence



Boot sequence



Boot sequence



Evaluation

- We care about
  - Correctness of the allocation
  - Maintainability of the code
  - Performance not the primary focus
- We used nine different real hardware systems for the evaluation

Evaluation

	C LOC
Register access	897
Data structure	1686
Resource management	706
ACPI	121
Interrupts	521
Miscellaneous	45
Total	3976

Table: LOC Linux

	C LOC	CLP LOC
Register access	235	
Data structure	817	31
Algorithm		224
ACPI	360	
Interrupts	660	28
Miscellaneous	109	
Total	2181	283

Table: LOC our approach

Evaluation

	C LOC
Register access	897
Data structure	1686
Resource management	706
ACPI	121
Interrupts	521
Miscellaneous	45
Total	3976

Table: LOC Linux

	C LOC	CLP LOC
Register access	235	
Data structure	817	31
Algorithm		224
ACPI	360	
Interrupts	660	28
Miscellaneous	109	
Total	2181	283

Table: LOC our approach

- Do not move a device:

Evaluation

	C LOC
Register access	897
Data structure	1686
Resource management	706
ACPI	121
Interrupts	521
Miscellaneous	45
Total	3976

Table: LOC Linux

	C LOC	CLP LOC
Register access	235	
Data structure	817	31
Algorithm		225
ACPI	360	
Interrupts	660	28
Miscellaneous	109	
Total	2181	284

Table: LOC our approach

- Do not move a device: call `keep_orig_addr()`

Evaluation

	C LOC
Register access	897
Data structure	1686
Resource management	706
ACPI	121
Interrupts	521
Miscellaneous	45
Total	3976

Table: LOC Linux

	C LOC	CLP LOC
Register access	235	
Data structure	817	31
Algorithm		225
ACPI	360	
Interrupts	660	28
Miscellaneous	109	
Total	2181	284

Table: LOC our approach

- Do not move a device: call `keep_orig_addr()`
- IOAPIC appears as BAR:

Evaluation

	C LOC
Register access	897
Data structure	1686
Resource management	706
ACPI	121
Interrupts	521
Miscellaneous	45
Total	3976

Table: LOC Linux

	C LOC	CLP LOC
Register access	235	
Data structure	817	31
Algorithm		239
ACPI	360	
Interrupts	660	28
Miscellaneous	109	
Total	2181	298

Table: LOC our approach

- Do not move a device: call `keep_orig_addr()`
- IOAPIC appears as BAR: implement `keep_ioapic_bars()`

Evaluation

	C LOC
Register access	897
Data structure	1686
Resource management	706
ACPI	121
Interrupts	521
Miscellaneous	45
Total	3976

Table: LOC Linux

	C LOC	CLP LOC
Register access	235	
Data structure	817	31
Algorithm		239
ACPI	360	
Interrupts	660	28
Miscellaneous	109	
Total	2181	298

Table: LOC our approach

- Do not move a device: call `keep_orig_addr()`
- IOAPIC appears as BAR: implement `keep_ioapic_bars()`
- Additional requirements to handle quirks easy to apply

Evaluation

Memory consumption and performance

- ECL^{i}PS^{e} is about 16242 LOCs of C
- Solver executable (statically linked): 1.5MB
- 600kB RAM disk
- 60MB dynamically allocated RAM buffers
- Execution time in the range of 2ms to 36ms

Conclusion

- PCI configuration in the real world is a hard, irregular problem
- Declarative languages
  - Tradeoff CPU cycles and memory footprint for simpler code
  - Facilitate handling quirks and other hardware bugs
- We think it is a promising approach for dealing with a large, diverse, and evolving hardware base

Download:
http://www.barrelfish.org

Changes to quirks.c

Kernel 2.6.36, 2005-2010

#commits	Year
26	2005
47	2006
49	2007
43	2008
42	2009
23	2010

Code examples

Keep original address

```
keep_orig_addr([], _, _, _, _, _, _).
keep_orig_addr([H|Tl], Cl, SubCl, PIf, Bs, Dv, Fn) :-
 (
  % if this is a device BAR...
  buselement(device,addr(Bs,Dv,Fn),BARNr,Base,_,_,_,_,_,_)
       = H,
  % and its device is in the required class...
  device(_,addr(Bs,Dv,Fn),_,_,Cl, SubCl, PIf,_),
  % lookup the original base address of the BAR
  bar(addr(Bs,Dv,Fn),BARNr,OrigBase,_,_,_,_) ->
    % constrain the Base to equal its original value
    Base $= OrigBase
  ; true
 ),
 % recurse on remaining devices
 keep_orig_addr(Tl, Cl, SubCl, PIf, Bs, Dv, Fn).
```

Discussion

Advantages

- Policy/mechanism separation
- Handle special cases completely in ECL^{i}PS^e
- General data entries
- Late-binding of algorithm
- Platform-independence

Discussion

Advantages

- Policy/mechanism separation
- Handle special cases completely in ECL^{i}PS^e
- General data entries
- Late-binding of algorithm
- Platform-independence

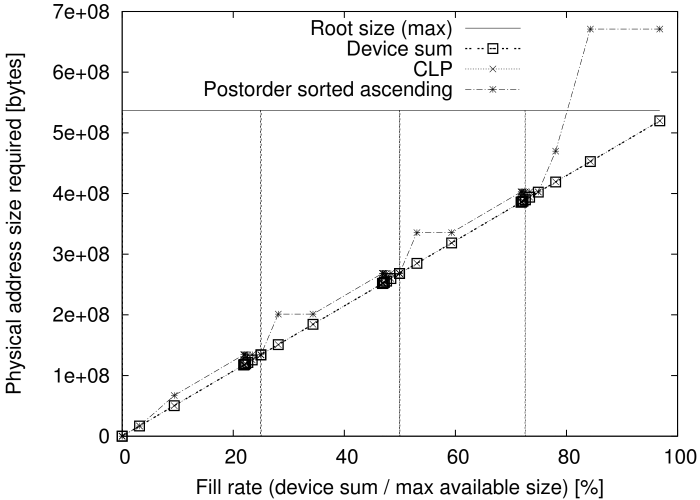
Disadvantages

- Increased resource usage
- Large code base
- Boot sequence
- Learning curve
- Need sometimes to understand how solver works

Space consumption

- artificial experiment
  - add more and more devices
  - sum of address space requests of all devices fill available range
  - monitor behaviour of postorder algorithm and CLP algorithm

Space consumption



Valid and invalid configurations

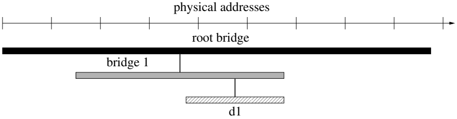
Valid configurations

Invalid configurations

Valid and invalid configurations

Valid configurations

Example 1

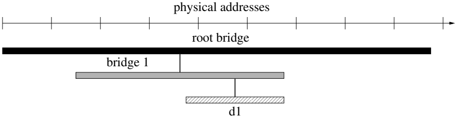


Invalid configurations

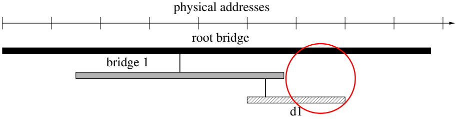
Valid and invalid configurations

Valid configurations

Example 1



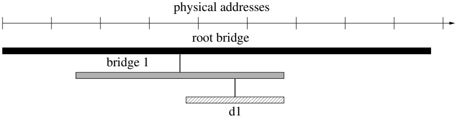
Invalid configurations



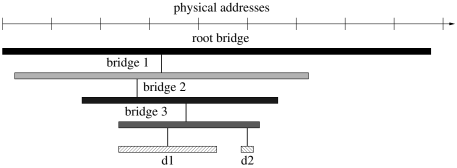
Valid and invalid configurations

Valid configurations

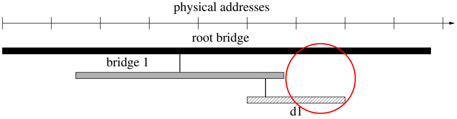
Example 1



Example 2



Invalid configurations

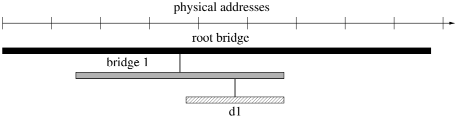


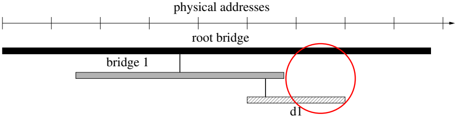
Valid and invalid configurations

Valid configurations

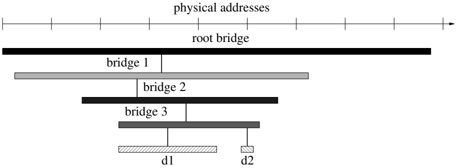
Invalid configurations

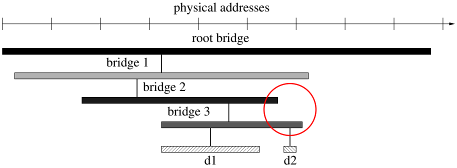
Example 1





Example 2



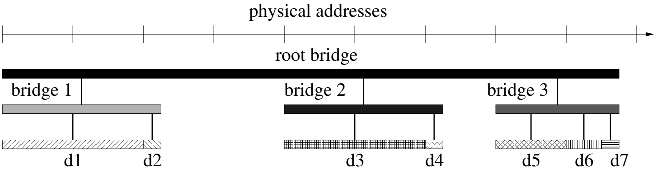


Why is it difficult?

- placement of child depends on placement of parent
- permutation of siblings possible at every level
- natural address alignment: big gaps possible
  - bad for resource utilization
  - good for hotplug
- fixed address requirements influence placing of parent bridges and siblings
- finding reasonable tree permutation is hard
  - changing order of bridges causes children to move as well
  - children can also be permuted

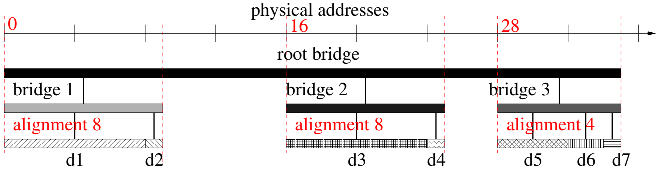
Why is it difficult?

Example



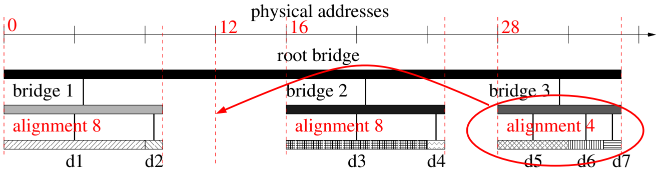
Why is it difficult?

Example



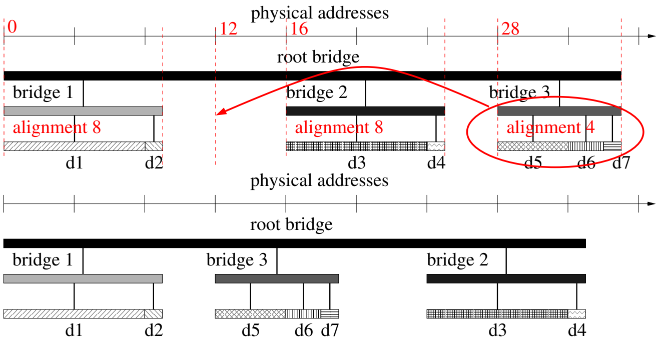
Why is it difficult?

Example



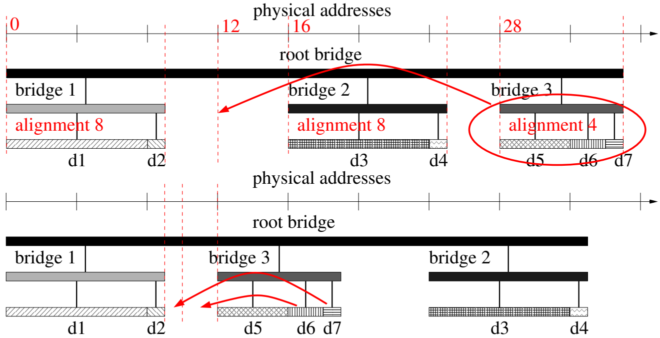
Why is it difficult?

Example



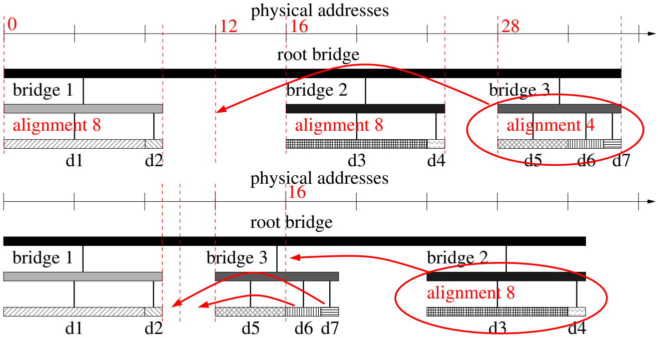
Why is it difficult?

Example



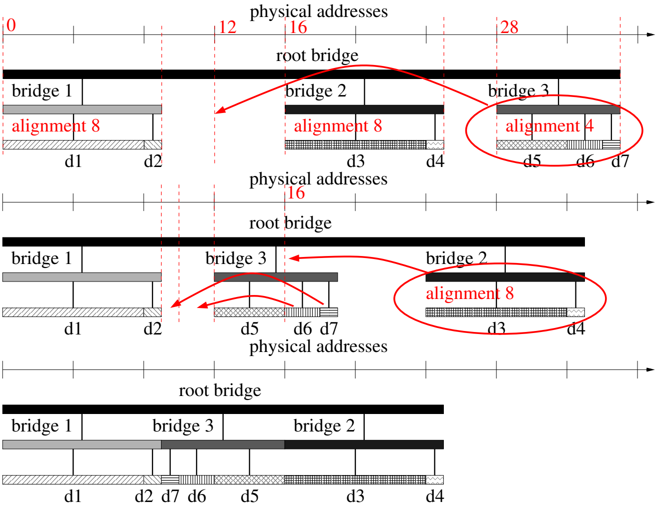
Why is it difficult?

Example



Why is it difficult?

Example



The Problem

- In theory, find a valid allocation of address ranges to devices, such that
  - All devices and bridges are configured
  - No overlapping of siblings occurs
  - Addresses are aligned to device specific boundaries
  - Children are within their parent bridge's address window
  - Complete PCI tree fits within available physical address space

The Problem

- In theory, find a valid allocation of address ranges to devices, such that
  - All devices and bridges are configured
  - No overlapping of siblings occurs
  - Addresses are aligned to device specific boundaries
  - Children are within their parent bridge’s address window
  - Complete PCI tree fits within available physical address space
- But in practice also
  - Certain devices can only have (partially) fixed addresses
  - Some bridges must be programmed with predefined values
  - Some physical regions have “holes” that can’t be used
  - “Quirks”

Implementation

- We use ECL^{i}PS^e-CLP to implement the algorithm
  - Prolog + constraint extensions
- We use a real system: Barrelfish
  - New operating system for heterogeneous manycore systems
  - Implemented from scratch → lots of freedom to try out ideas
- Implementation done in the system knowledge base (SKB)
  - User-space service containing ECL^{i}PS^e
  - Contains data base with hardware facts in Prolog form
  - Uses RAM disk to access ECL^{i}PS^e code and is self-contained

Boot sequence in Barrelfish

CPU
PCI bus

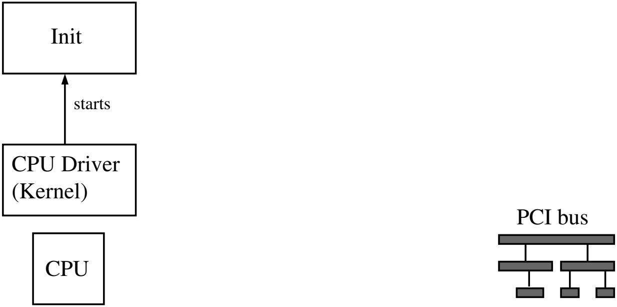
Boot sequence in Barrelfish

CPU Driver (Kernel)

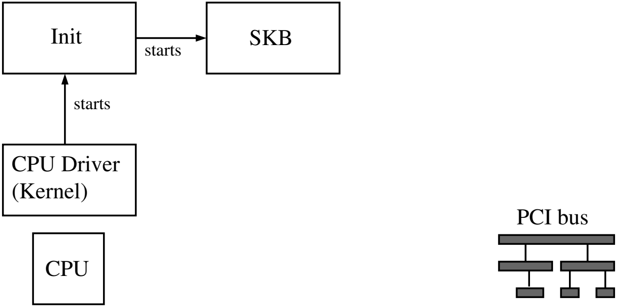
CPU

PCI bus

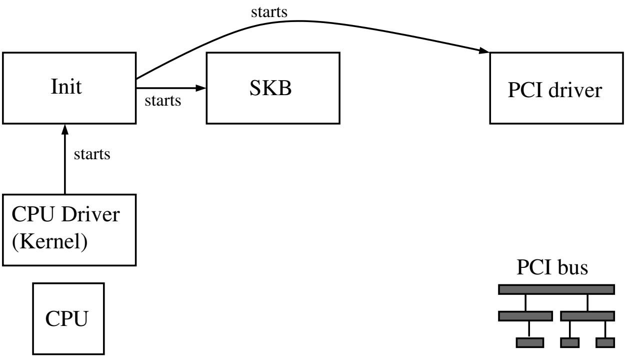
Boot sequence in Barrelfish



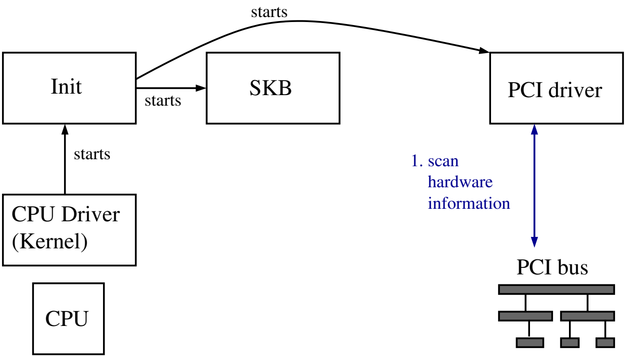
Boot sequence in Barrelfish



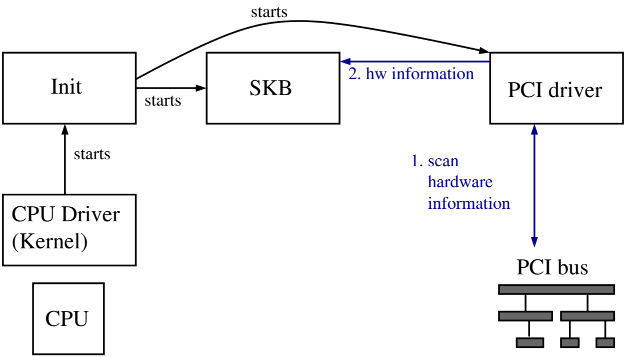
Boot sequence in Barrelfish



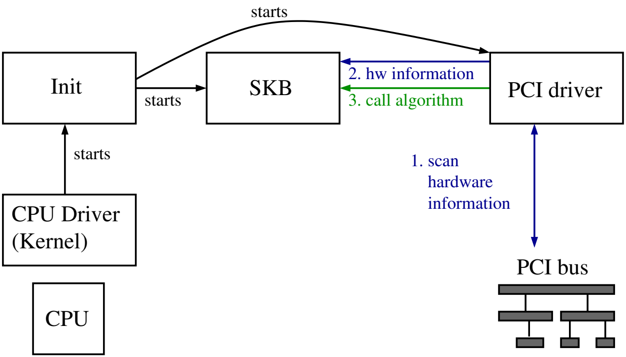
Boot sequence in Barrelfish



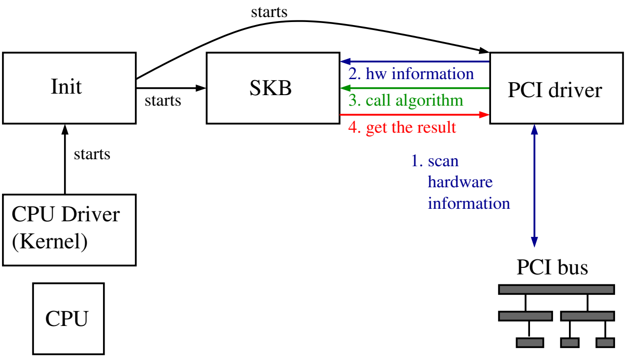
Boot sequence in Barrelfish



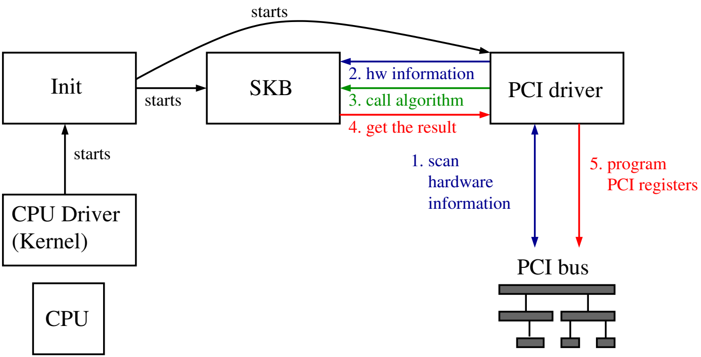
Boot sequence in Barrelfish



Boot sequence in Barrelfish



Boot sequence in Barrelfish


starts
Init
starts
SKB
2. hw information
3. call algorithm
4. get the result
PCI driver
starts
1. scan hardware information
5. program PCI registers
CPU Driver (Kernel)
PCI bus
CPU