Practical Probability
Applying pGCL to Lattice Scheduling

David Cock

25 July 2013

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Overview

In this talk, I present a **verified lattice scheduler**, that eliminates leakage via a shared cache, while guaranteeing non-starvation. In addition, this work:

- Applies our existing pGCL package for Isaelle.
- Presents a multilevel probabilistic refinement proof.
- Integrates with the seL4 proof.

Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary

- Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.

- Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.
- Any output from an agent clear for A is tagged A, likewise for B.

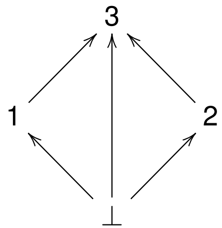
- Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.
- Any output from an agent clear for A is tagged A, likewise for B.
- There are four possible clearances: A, B, A and B, and nothing. These are **domains**.

- Consider a system with two classification tags: A and B. Information tagged with A may only be seen by an agent cleared to see A, likewise for B.
- Any output from an agent clear for A is tagged A, likewise for B.
- There are four possible clearances: A, B, A and B, and nothing. These are **domains**.
- The who-may-talk-to-whom order is a lattice:

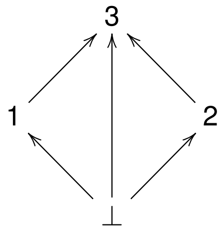
$$
\begin{tikzcd}
 & \{A, B\} & \\
\{A\} \arrow[ur] & & \{B\} \arrow[ul] \\
 & \{\,\} \arrow[ul] \arrow[ur] \arrow[uu] &
\end{tikzcd}
$$

$$
\begin{tikzcd}
 & 3 & \\
1 \arrow[ur] & & 2 \arrow[ul] \\
 & \perp \arrow[ul] \arrow[ur] \arrow[uu] &
\end{tikzcd}
$$

- For brevity, label the domains and then forget the sets.



- For brevity, label the domains and then forget the sets.
- Enforcing rules for explicit communication in such a system is a well-studied problem.

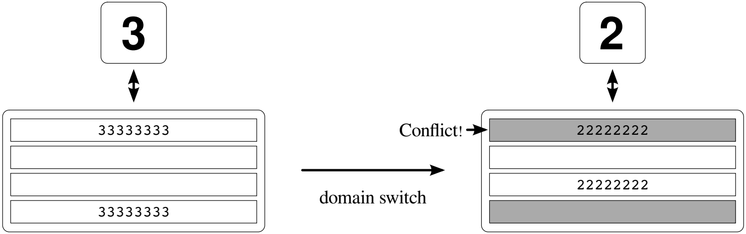


- For brevity, label the domains and then forget the sets.
- Enforcing rules for explicit communication in such a system is a well-studied problem.
- **Implicit** communication is harder.

$$\begin{tikzcd} & 3 & \\ 1 \arrow[ur] & & 2 \arrow[ul] \\ & \perp \arrow[ul] \arrow[ur] \arrow[uu] & \end{tikzcd}$$

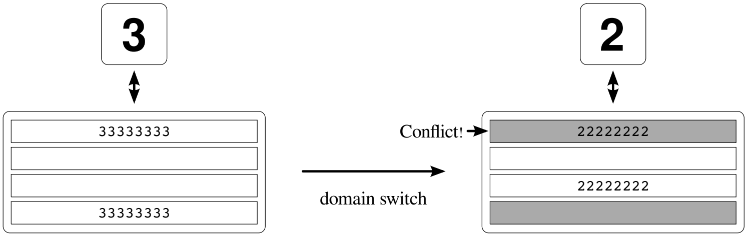
- For brevity, label the domains and then forget the sets.
- Enforcing rules for explicit communication in such a system is a well-studied problem.
- **Implicit** communication is harder.
- We're specifically concerned with covert channels due to sharing hardware.

The Cache Channel



Even if two domains are unable to communicate, they leave detectable traces in the machine state.

The Cache Channel



Even if two domains are unable to communicate, they leave detectable traces in the machine state.
For example, 2 cannot read 3's cache lines, but it can infer where they are, by timing its own memory accesses.

The Cache Channel

How do we mitigate this channel?

The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime

The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime . . . expensive!

The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime . . . expensive!
- We don't **need** to flush when transitioning up.

The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime . . . expensive!
- We don't **need** to flush when transitioning up.
- Transition up as long as possible. . .

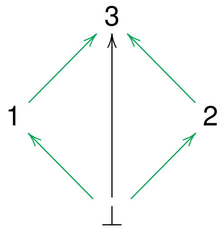
The Cache Channel

How do we mitigate this channel?

- We could flush the cache everytime . . . expensive!
- We don't **need** to flush when transitioning up.
- Transition up as long as possible. . . then flush and start again.

This is **Lattice Scheduling**

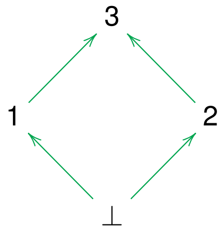
$$
\begin{tikzcd}
 & 3 & \\
1 \arrow[ur] & & 2 \arrow[ul] \\
 & \bot \arrow[ul] \arrow[ur] \arrow[uu] &
\end{tikzcd}
$$



- The schedule relation S, is a subset of the up transitions.

$$
\begin{tikzcd}
 & 3 & \\
1 \arrow[ur] & & 2 \arrow[ul] \\
 & \perp \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

- The schedule relation S, is a subset of the up transitions.



- The schedule relation S, is a subset of the up transitions.
- This schedule is incomplete: There's no way to leave 3.

$$
\begin{tikzcd}
 & 3 & \\
1 \arrow[ur] & & 2 \arrow[ul] \\
 & \perp \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

- The schedule relation S, is a subset of the up transitions.
- This schedule is incomplete: There's no way to leave 3.
- We must add downward transitions, but how?

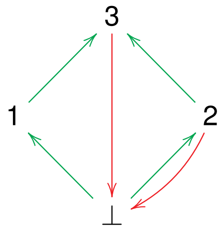
$$\begin{tikzcd}
 & 3 & \\
1 \arrow[ur] & & 2 \arrow[ul] \\
 & \perp \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

- Designate a downgrader, $\perp$.

$$
\begin{tikzcd}
& 3 & \\
1 \arrow[ur] & & 2 \arrow[ul] \\
& \perp \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

- Designate a downgrader, $\perp$.
- The downgrader clears the cache.

NICTA



- Designate a downgrader, $\perp$.
- The downgrader clears the cache.

Lemma (Downgrading)

If S allows a downward transition, it is to the downgrader, $\perp$:

$$\frac{(c, n) \in S \qquad \textit{clearance } c \not\subseteq \textit{clearance } n}{n = \perp}$$

The Lattice Scheduler

We'll verify a scheduler written in pGCL, an imperative, probabilistic language:

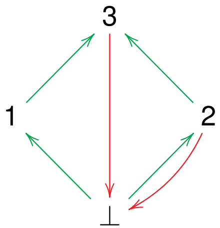
$$\textbf{record}\ \text{stateA} = \text{current\_domain} :: \textit{dom\_id}$$

$$\text{scheduleS} =$$

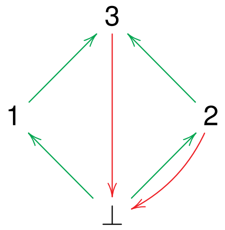
$$\quad c \text{ is current\_domain in}$$

$$\quad \text{current\_domain} :\in (\lambda\_.\ \{n.\ (c, n) \in S\})$$

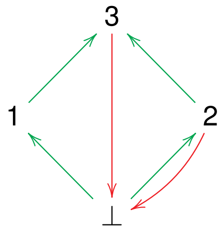
This program selects a new domain nondeterministically from among those with a valid transition from the current.



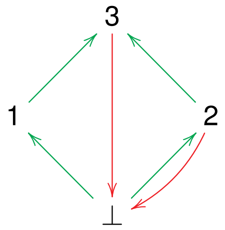
- We want to **refine** this to a realistic implementation.



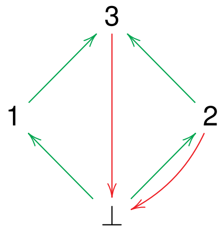
- We want to **refine** this to a realistic implementation.
- The refinement may produce any trace permitted here.



- We want to **refine** this to a realistic implementation.
- The refinement may produce any trace permitted here.
- For example: $\perp, 2, \perp, 2, \ldots$.



- We want to **refine** this to a realistic implementation.
- The refinement may produce any trace permitted here.
- For example: $\perp, 2, \perp, 2, \ldots$.
- The specification permits starvation.



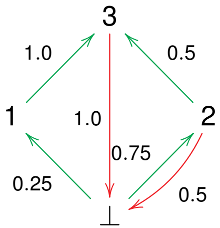
- We want to **refine** this to a realistic implementation.
- The refinement may produce any trace permitted here.
- For example: $\perp, 2, \perp, 2, \ldots$.
- The specification permits starvation.
- Randomisation gives us a neat solution.

Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary

NICTA

Random Transitions



- Assign a probability to each transition such that $T\ (c, n) > 0$ only if $(c, n) \in S$.
- Outgoing probabilities sum to 1 (or less).
- The previous trace now has probability 0!

A Probabilistic Scheduler

scheduleT $=$

c is current_domain in

current_domain $:\in (\lambda\_.\ \{\bot, 1, 2, 3\}$ at $(\lambda\_\ n.\ T\ (c, n))$

A Probabilistic Scheduler

$$\text{scheduleT} =$$
$$\quad c \text{ is current\_domain in}$$
$$\quad \text{current\_domain} :\in (\lambda\_.\ \{\bot, 1, 2, 3\} \text{ at } (\lambda\_\ n.\ T\ (c, n))$$

Lemma (Non-starvation)

Taking at least 8 steps from any *initial domain, we reach* any *final domain with non-zero probability:*

$$\forall s.\ 0 < wp\ scheduleT^{8+n}\ (in\_dom\ d_f)\ s$$

Note that predicates (expectations) in pGCL are real-valued.

Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.

Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment ($\vdash$) generalises ($\vDash$), which is really just $\leq$:

$$\text{False} \rightarrow \text{True} \qquad 0 \leq 1$$

$$\lambda x.\ \text{False} \vdash \lambda x.\ \text{True} \qquad \lambda x.\ 0 \vDash \lambda x.\ 1$$

Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment ($\vdash$) generalises ($\vDash$), which is really just $\leq$:

$$\text{False} \rightarrow \text{True} \qquad\qquad 0 \leq 1$$

$$\lambda x.\ \text{False} \vdash \lambda x.\ \text{True} \qquad\qquad \lambda x.\ 0 \vDash \lambda x.\ 1$$

- Predicates are lifted to **expectations**:

$$«P» = \lambda x.\ \text{if } P\ x \text{ then } 1 \text{ else } 0$$

Refinement in pGCL

What about downgrading, does it still hold? We show this using refinement, but first some notes on pGCL:

- pGCL generalises Boolean logic with real values: True is 1, False is 0.
- Entailment ($\vdash$) generalises ($\vDash$), which is really just $\leq$:

$$\text{False} \rightarrow \text{True} \qquad\qquad 0 \leq 1$$

$$\lambda x.\ \text{False} \vdash \lambda x.\ \text{True} \qquad\qquad \lambda x.\ 0 \vDash \lambda x.\ 1$$

- Predicates are lifted to **expectations**:

$$«P» = \lambda x.\ \text{if } P\ x \text{ then } 1 \text{ else } 0$$

- We reason about **weakest-preexpectations**:

$$Q \vDash \text{wp prog } R$$

Refinement in pGCL

pGCL refinement has the usual properties:

Refinement in pGCL

pGCL refinement has the usual properties:

Definition

Program b *refines* program a, written $a \sqsubseteq b$, exactly when all expectation-entailments on a also hold on b:

$$\frac{P \vDash \text{wp } a\ Q}{P \vDash \text{wp } b\ Q}$$

Refinement in pGCL

pGCL refinement has the usual properties:

Definition

Program b *refines* program a, written $a \sqsubseteq b$, exactly when all expectation-entailments on a also hold on b:

$$\frac{P \vDash \mathrm{wp}\ a\ Q}{P \vDash \mathrm{wp}\ b\ Q}$$

Lemma

The transition scheduler refines the lattice scheduler:

$$\mathit{scheduleS} \sqsubseteq \mathit{scheduleT}$$

First Refinement

$$\begin{tikzcd} \textsc{Downgrading} \arrow[r, dotted] & \text{scheduleS} \arrow[d, "\sqsubseteq"] \\ & \text{scheduleT} \end{tikzcd}$$

- Downgrading is preserved by refinement,

First Refinement

$$
\begin{tikzcd}
\textsc{Downgrading} \arrow[r, dotted] \arrow[dr, dotted] & \text{scheduleS} \arrow[d, "\sqsubseteq"] \\
 & \text{scheduleT}
\end{tikzcd}
$$

- Downgrading is preserved by refinement, and therefore holds for scheduleT.

NICTA
Lattice Scheduling
The Probabilistic Scheduler
Refinement
Lottery Scheduling
Data Refinement
seL4 Integration
Non-Leakage
Summary

First Refinement

$$\begin{tikzcd} \textsc{Downgrading} \arrow[r, dotted] \arrow[dr, dotted] & \text{scheduleS} \arrow[d, "\sqsubseteq"] \\ \textsc{Non-Starvation} \arrow[r, dotted] & \text{scheduleT} \end{tikzcd}$$

- Downgrading is preserved by refinement, and therefore holds for scheduleT.
- Non-starvation holds **only** for scheduleT.

Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary

The Lottery Scheduler

Our scheduler is so far very abstract. The next step is to **implement** the randomisation. We use a lottery:

- We only need a uniform random choice from $\mathbb{Z}_{32}$.
- Each option is assigned some number, x, of tickets.
- The chance of winning is $\frac{x}{2^{32}}$.
- We need to assume that the **lottery relation** holds:

$$T\ (c, n) = 2^{-32} \|\{t.\ \text{lottery}\ (\text{domains}\ s\ c)\ t = n\}\|$$

- Different state spaces: need more than simple refinement.

The Lottery Scheduler

$$\textbf{record}\ \text{domain} = \text{lottery} :: 32\ \text{word} \Rightarrow \mathit{dom\_id}$$

$$\begin{aligned}\textbf{record}\ \text{stateC} = {} & \text{current\_domain} :: \mathit{dom\_id} \\ & \text{domains} :: \mathit{dom\_id} \Rightarrow \text{domain}\end{aligned}$$

$$\begin{aligned}\text{scheduleM}\ t = \text{do}\ & c \leftarrow \text{gets current\_domain} \\ & \mathit{dl} \leftarrow \text{gets domains} \\ & \text{let}\ n = \text{lottery}\ (\mathit{dl}\ c)\ t\ \text{in} \\ & \text{modify}\ (\lambda s.\ s(\!|\text{current\_domain} := n|\!)) \\ \text{od} &\end{aligned}$$

$$\begin{aligned}\text{scheduleC} = {} & t\ \text{from}\ (\lambda s.\ \text{UNIV})\ \text{at}\ 2^{-32}\ \text{in} \\ & \text{Exec}\ (\text{scheduleM}\ t)\end{aligned}$$

Data Refinement

Definition (Probabilistic Data Refinement)

Program b, on state type σ, refines program a, state τ, given precondition $G : \sigma \to$ Bool and under projection $\theta : \sigma \to \tau$, written $a \sqsubseteq_{G,\theta} b$, exactly when any expectation entailment on a implies the same for b, on the projected state and with a guarded pre-expectation:

$$\frac{P \vDash \text{wp}\ a\ Q}{«G» \mathbin{\&\&} (P \circ \theta) \vDash \text{wp}\ b\ (Q \circ \theta)}$$

$$a \mathbin{\&\&} b = \max\ (a + b - 1)\ 0$$

Correspondence

Definition (Probabilistic Correspondence)

Programs *a* and *b* are said to be in *probabilistic correspondence*, pcorres θ *G a b*, given condition *G* and under projection θ if, for any post-expectation *Q*, the guarded pre-expectations coincide:

$$\text{«}G\text{»} \,\&\&\, (\text{wp}\ a\ Q \circ \theta) = \text{«}G\text{»} \,\&\&\, \text{wp}\ b\ (Q \circ \theta)$$

Correspondence

Definition (Probabilistic Correspondence)

Programs *a* and *b* are said to be in *probabilistic correspondence*, pcorres θ *G a b*, given condition *G* and under projection θ if, for any post-expectation *Q*, the guarded pre-expectations coincide:

$$\text{«}G\text{»} \mathbin{\&\&} (\mathrm{wp}\ a\ Q \circ \theta) = \text{«}G\text{»} \mathbin{\&\&} \mathrm{wp}\ b\ (Q \circ \theta)$$

Lemma

The specifications scheduleT and scheduleC correspond given condition LR and under projection ϕ:

$$\mathit{pcorres}\ \phi\ \mathit{LR}\ \mathit{scheduleT}\ \mathit{scheduleC}$$

Second Refinement

$$\begin{tikzcd}
\textsc{Downgrading} \arrow[r, dotted] & \text{scheduleS} \arrow[d] \\
\textsc{Non-Starvation} \arrow[r, dotted] & \text{scheduleT}
\end{tikzcd}$$

Second Refinement

$$\begin{tikzcd} \textsc{Downgrading} \arrow[r, dotted] & \text{scheduleS} \arrow[d] \\ \textsc{Non-Starvation} \arrow[r, dotted] & \text{scheduleT} \arrow[d, Rightarrow, "\phi\text{,LR}"] \\ & \text{scheduleC} \end{tikzcd}$$

- The double arrow represents correspondence.

Second Refinement

$$
\begin{tikzcd}
\textsc{Downgrading} \arrow[r, dotted] & \text{scheduleS} \arrow[d] \\
\textsc{Non-Starvation} \arrow[r, dotted] & \text{scheduleT} \arrow[d, Rightarrow, "\phi\text{,LR}"] \\
 & \text{scheduleC}
\end{tikzcd}
$$

- The double arrow represents correspondence.
- Correspondence composes with refinement.

Second Refinement

$$
\begin{tikzcd}
\textsc{Downgrading} \arrow[r, dotted] & \text{scheduleS} \arrow[d] \\
\textsc{Non-Starvation} \arrow[r, dotted] & \text{scheduleT} \arrow[d, Rightarrow, "\phi\text{,LR}"] \\
 & \text{scheduleC}
\end{tikzcd}
$$

- The double arrow represents correspondence.
- Correspondence composes with refinement.
- Downgrading and non-starvation are preserved.

Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary

The Nondeterministic State Monad

- The seL4 specification is written using a **nondeterministic state monad**.
- We can embed this cleanly into pGCL.

The Nondeterministic State Monad

- The seL4 specification is written using a **nondeterministic state monad**.
- We can embed this cleanly into pGCL.
- In fact, we just used it: scheduleM and Exec.

The Nondeterministic State Monad

- The seL4 specification is written using a **nondeterministic state monad**.
- We can embed this cleanly into pGCL.
- In fact, we just used it: scheduleM and Exec.
- L4.verified used a particular notion of nondeterministic correspondence.

The Nondeterministic State Monad

- The seL4 specification is written using a **nondeterministic state monad**.
- We can embed this cleanly into pGCL.
- In fact, we just used it: scheduleM and Exec.
- L4.verified used a particular notion of nondeterministic correspondence.
- We know how to lift these results, probabilistically:

Lemma (Lifting Correspondence)

Given correspondence between M and M′: with

$$\mathit{corres\_underlying}\ \{(s, s').\ s = \phi\ s'\}\ \mathit{True}\ \mathit{rrel}\ G\ (G \circ \phi)\ M\ M'$$

and standard side-conditions:

$$\mathit{no\_fail}\ G\ M \qquad \mathit{empty\_fail}\ M \qquad \mathit{empty\_fail}\ M'$$

and that M is deterministic on the image of the projection,

$$\forall s.\ \exists (r, s').\ M\ (\phi\ s) = \{(\mathit{False}, (r, s'))\}$$

then we have probabilistic *correspondence:*

$$\mathit{pcorres}\ \phi\ (G \circ \phi)\ (\mathit{Exec}\ M)\ (\mathit{Exec}\ M')$$

Lemma

If the kernel preserves the lottery relation,

$$\{\!|LR|\!\}\ \mathit{stepKernel}\ \{\!|\lambda\_.\ LR|\!\}$$

and the current domain,

$$\{\!|\lambda s.\ CD\ s = d|\!\}\ \mathit{stepKernel}\ \{\!|\lambda\_\ s.\ CD\ s = d|\!\}$$

and is total,

$$\mathit{no\_fail}\ \top\ \mathit{stepKernel} \quad \mathit{empty\_fail}\ \mathit{stepKernel}$$

then with the concrete scheduler, it refines the transition scheduler:

$$\mathit{scheduleT} \sqsubseteq_{LR,\phi} \mathit{stepKernel};;\mathit{scheduleC}$$

Composed Refinement

$$
\begin{tikzcd}
& \textsc{Downgrading} \arrow[r, dotted] & \text{scheduleS} \arrow[d] \\
& \textsc{Non-Starvation} \arrow[r, dotted] & \text{scheduleT} \arrow[d, Rightarrow, "\phi\text{,LR}"] \\
\text{callKernelD} = = = \text{stepKernel; ; scheduleC} \arrow[d, Rightarrow] & & \text{scheduleC} \arrow[ll, Rightarrow, "\phi\text{,LR}"'] \\
\text{callKernelH} \arrow[d, Rightarrow] & & \\
\text{callKernelC} & &
\end{tikzcd}
$$

Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary

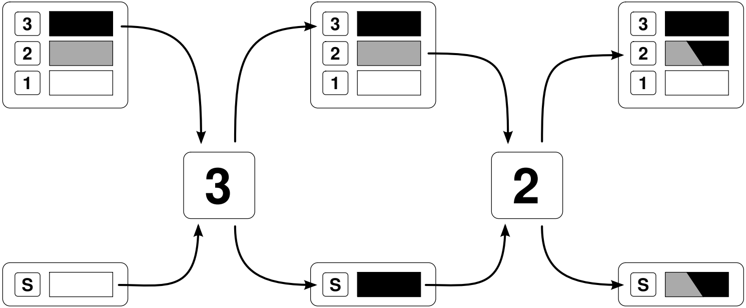
Hardware Model

Ultimately, we want to know that our scheduler eliminates leakage via the cache. We append a machine model:

$$\textbf{record}\ (\mathit{sh}, \mathit{pr})\ \text{machine} = \text{private} :: \mathit{dom\_id} \Rightarrow \mathit{pr}$$
$$\text{shared} :: \mathit{sh}$$

- A private state per domain.
- A shared state between domains (the cache).
- Domains are underspecified, but may only update their own private state and the shared state.

Leakage via Shared State



- Propagating taint takes at least 2 steps.
- A single-step policy isn't enough.

Lemma (Non-leakage)

If the clearance of domain h is not entirely contained within that of domain l,

$$\textit{clearance } h \not\subseteq \textit{clearance } l$$

then any function of the state after execution, which depends only on elements within l's clearance,

$$Q \circ \textit{mask } l$$

is invariant under modifications to h's private state (as represented by replace):

$$\begin{aligned} & wp\ (runDom;;scheduleT)^n\ (Q \circ mask) = \\ & \quad (wp\ (runDom;;scheduleT)^n\ (Q \circ mask)) \circ (replace\ h\ p) \end{aligned}$$

Final Refinement

$$
\begin{tikzcd}
\textsc{Downgrading} \arrow[r, dotted] & \mathsf{scheduleS} \arrow[d] & \textsc{Non-Leakage} \arrow[d, dotted] \\
\textsc{Non-Starvation} \arrow[r, dotted] & \mathsf{scheduleT} \arrow[r, Rightarrow, "{\psi,\mathsf{LR}}"'] \arrow[d, Rightarrow, "{\phi,\mathsf{LR}}"] & \mathsf{runDom\ ;;\ scheduleT} \arrow[d, Rightarrow, "{\phi,\mathsf{LR}}"] \\
\mathsf{stepKernel\ ;;\ scheduleC} & \mathsf{scheduleC} \arrow[l, Rightarrow, "{\phi,\mathsf{LR}}"] \arrow[r, Rightarrow, "{\psi,\mathsf{LR}}"'] & \mathsf{runDom\ ;;\ scheduleC}
\end{tikzcd}
$$

- We've have non-leakage for the probabilistic scheduler (scheduleT), and it is preserved by refinement.
- We now have all 3 properties for the concrete implementation.

Outline

- Lattice Scheduling
- The Probabilistic Scheduler
  - Refinement
- Lottery Scheduling
  - Data Refinement
- seL4 Integration
- Non-Leakage
- Summary

NICTA

What The Message?

What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.

What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists

What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists — pGCL for Isabelle available from:
  `http://www.cse.unsw.edu.au/~davec/pGCL/`
  Will also to be submitted to AFP.

What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists — pGCL for Isabelle available from:
  `http://www.cse.unsw.edu.au/~davec/pGCL/`
  Will also to be submitted to AFP.
- Some problems in security are **unavoidably** probabilistic.

What The Message?

- Probabilistic programs need not be harder to verify than traditional ones.
- Good tool support now exists — pGCL for Isabelle available from:
  `http://www.cse.unsw.edu.au/~davec/pGCL/`
  Will also to be submitted to AFP.
- Some problems in security are **unavoidably** probabilistic.
- Probabilistic results can compose well with large existing proofs.

NICTA

Questions?