Concurrent Programming via Access Permissions

Sven Stork

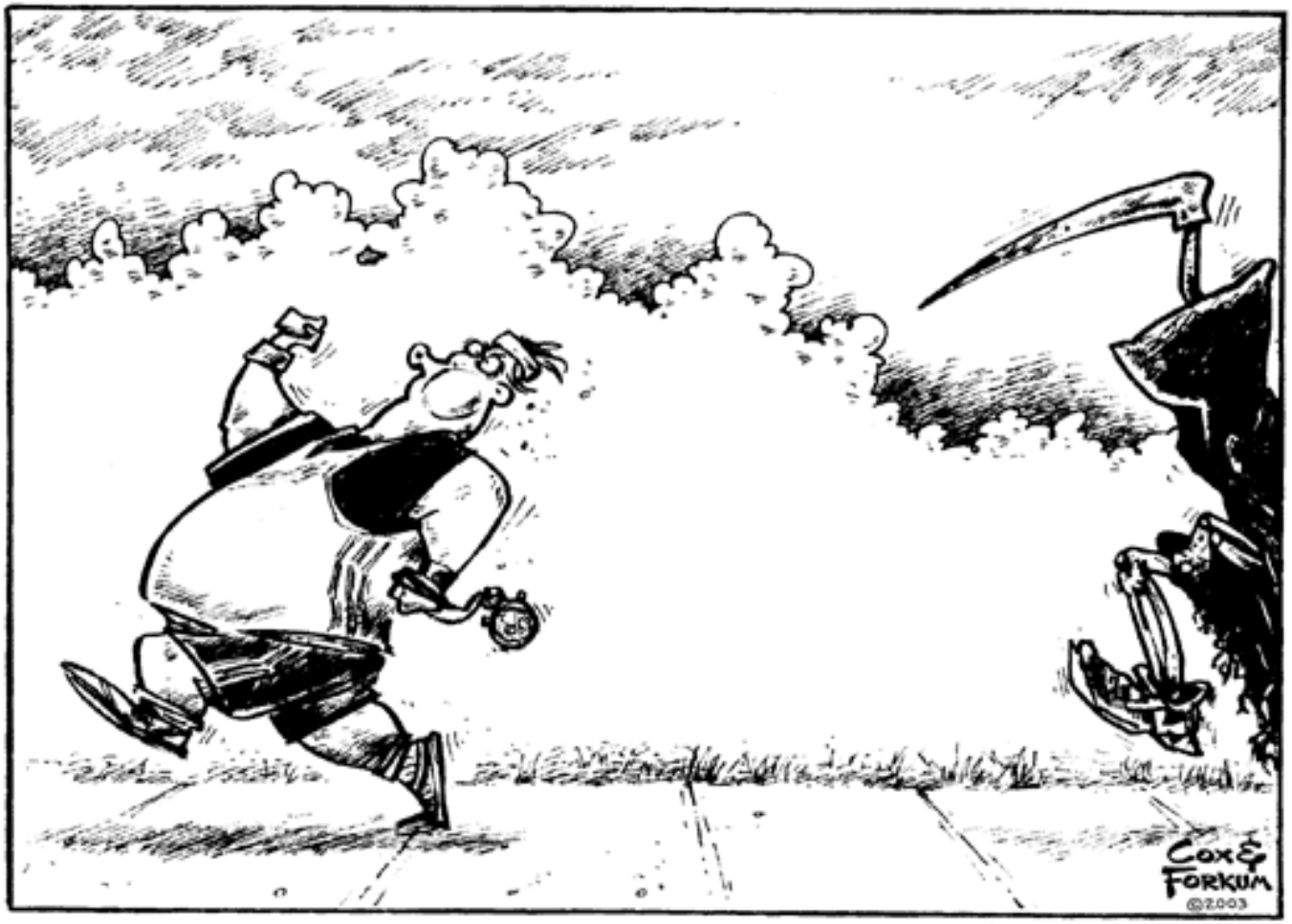
Carnegie Mellon University
svens@cs.cmu.edu
Supervisor: Jonathan Aldrich

University of Coimbra
stork@dei.uc.pt
Supervisor: Paulo Marques

October 7, 2009

Information and Communication Technologies Institute
Carnegie Mellon | PORTUGAL
AN INTERNATIONAL PARTNERSHIP

Motivation

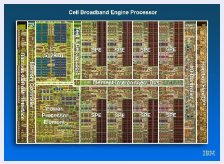
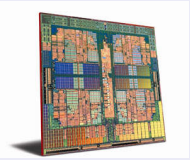


www.CoxAndForkum.com

The Problem

Era of Concurrency is upon us!

- concurrency is (basically) everywhere


- major paradigm shift in technology
- **"The free lunch is over"** (Sutter, 2005)
- programs need to become concurrent to achieve higher performance

The Problem

What is the problem with concurrent programming?

explicit concurrent programming

- **default** in mainstream object oriented languages
- **"manual"** management of concurrent (e.g., threads, locks, ...)
- concurrent issues **additionally** to sequential problems
- **complex** and **error prone** reasoning about ordering constraints

implicit concurrent programming

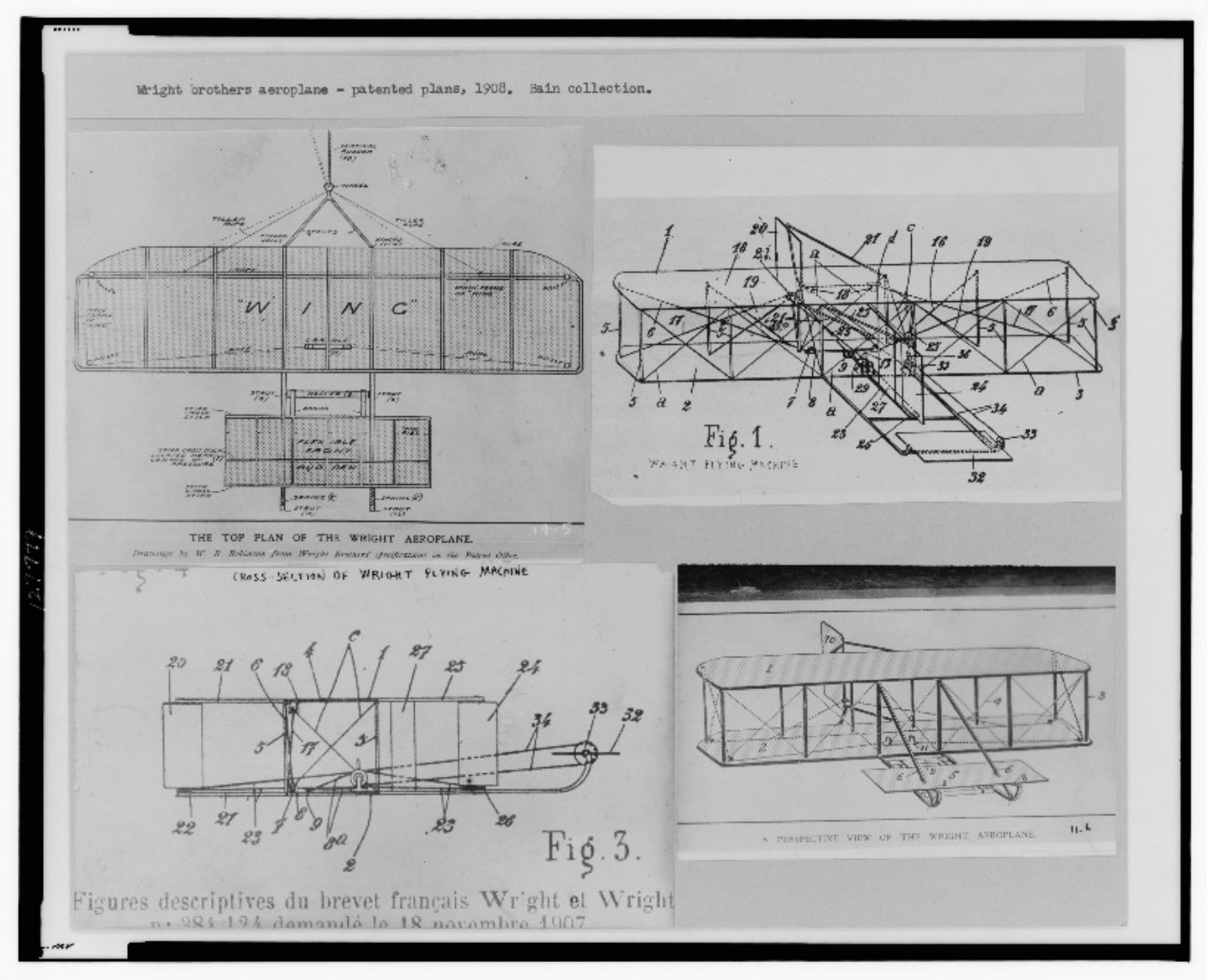
- tell the system **what to do** and **not how to do it**
- functional programming languages (**no state**)
- works well for certain domains (e.g., data parallel)
  - does not work for general purpose tasks (e.g., web server)

The Problem

Research Questions

- Can you have an implicit concurrency model for a general purpose object oriented programming language ?
  - ⇝ Can you have a language that **automatically infers concurrency information**?
  - ⇝ Can you have a language that **protect against common pitfalls** (e.g., race conditions)?
- What are the requirements for a runtime systems, to execute such programs efficiently ?
  - ⇝ How to use the **mixture fine - coarse grain concurrency information to execute a program efficiently**?
- How much concurrency can be extracted via such a high level abstraction ?
  - ⇝ How much **performance improvements** can **real world applications** achieve ?

Approach


Wright brothers aeroplane - patented plans, 1908. Bain collection.
"W I N C"
THE TOP PLAN OF THE WRIGHT AEROPLANE.
Fig. 1.
CROSS-SECTION OF WRIGHT FLYING MACHINE
Fig. 3.
Figures descriptives du brevet français Wright et Wright

Approach

Research Question

- Can you have an implicit concurrency model for a general purpose object oriented programming language ?
  - ⇝ Can you have a language that **automatically infers concurrency information**?
  - ⇝ Can you have a language that **protect against common pitfalls** (e.g., race conditions)?

Our Approach

- Problem: lack of **exact dependency information** caused by **aliasing**
- Remedy: use **access permissions**
  1. provide **aliasing** and **data access** information
  2. calculate **more precise dependencies**
  3. infer **concurrency information** obeying dependencies (**dataflow graph**)
  4. check against **correct usage**

Approach

Research Question

- Can you have an implicit concurrency model for a general purpose object oriented programming language ?
  - ⇝ Can you have a language that **automatically infers con**
    information?
  - ⇝ Can you have a language that **prot**
    race conditions)?

- programming paradigm : **concurrency by default**
- programming language : **ÆMINIUM**

Our Ap

- **dependency information** caused by **aliasing**
- use **access permissions**

1. provide **aliasing** and **data access** information
2. calculate **more precise dependencies**
3. infer **concurrency information** obeying dependencies (**dataflow graph**)
4. check against **correct usage**

Access Permissions

What kind of Access Permissions do we use ?

unique

- there is only one reference to the object
- **exclusive access**
- no synchronization required

immutable

- the object **cannot** be modified through an immutable reference
- there might be several alias reference to the object, but **all** of them are **immutable**
- no synchronization required

shared

- the object **can** be modified through an shared reference
- there might be several alias reference to the object, but **all** of them are **shared**
- **access** to shared objects requires synchronization

Access Permissions

What kind of Access Permissions do we use ?

unique

- there is only one reference to the object
- **exclusive access**
- no synchronization required

immutable

- the object **cannot** be modified through an immutable reference
- there might be several alias reference to the object, but **all** of them are **immutable**
- no synchronization required

shared

- the object **can** be modified [illegible] reference [illegible] alias reference to the object, [illegible] of them are **shared**
- **access** to shared objects requires synchronization

synchronization ENFORCED

Access Permissions

How to extract concurrency out of access permissions?

- **Access Permissions** encode **data access** and **aliasing information**
  - **automatically splitting/joining** of permissions
    e.g., unique $\Leftarrow\Rrightarrow$ immutable $\otimes$ immutable
    e.g., unique $\Leftarrow\Rrightarrow$ shared $\otimes$ shared
  - use **linear logic** for management access permissions
- **"reverse"** this approach and **infer** which operations can be executed concurrently

We define that ...

- pieces of code **can** be executed concurrently, **iff** they dependent on

  immutable permissions $\rightsquigarrow$ only read operations
  shared permissions $\rightsquigarrow$ access must synchronized

- we propose **data groups** for specifying additional ordering

Access Permissions

Program - main

```
void main() {
  Collection c = readData()
  printCollection(c)
  Statistics s = compStats(c)
  Dependencies d = compDeps(c)
  removeDuplicates(c)
  printCollection(c)
  ...
}
```

Access Permissions

Program - main

```
void main() {
  Collection c = readData()
  printCollection(c)
  Statistics s = compStats(c)
  Dependencies d = compDeps(c)
  removeDuplicates(c)
  printCollection(c)
  ...
}
```

Question

- Is there concurrency in this program?

Access Permissions

Program - Types

Collection readData()
: () → Collection

void removeDuplicates(Collection c)
: Collection → ()

void printCollection(Collection c)
: Collection → ()

Dependencies compDeps(Collection c)
: Collection → Dependencies

Statistics compStats(Collection c)
: Collection → Statistics

Access Permissions

Program - Types

Collection readData()
: () → Collection

void re
: Coll

void pr
: Collection ()

Dependencies compDeps(Collection c)
: Collection → Dependencies

Statistics compStats(Collection c)
: Collection → Statistics

Question

- typing specifies input/output of behavior
- Q: Can we run `compDeps` and `compStats` ?
- A: Depends if there are dependencies between them

Access Permissions

Program - Access Permissions

Collection readData()
: **unit** $\Mapsto$ **unique**(**result**)

void removeDuplicates(Collection c)
: **unique**(c) $\Mapsto$ **unique**(c)

void printCollection(Collection c)
: **immutable**(c) $\Mapsto$ **immutable**(c)

Dependencies compDeps(Collection c)
: **immutable**(c) $\Mapsto$ **immutable**(c), **unique**(**result**)

Statistics compStats(Collection c)
: **immutable**(c) $\Mapsto$ **immutable**(c), **unique**(**result**)

Access Permissions

Program - Access Permissions

Collection readData()
: **unit** ⇒ **unique**(**result**)

void re
: **uniq**

void pr
: **imm**

Depen
: **immutable**(c) ⇒ **immutable**(c), **unique**(**result**)

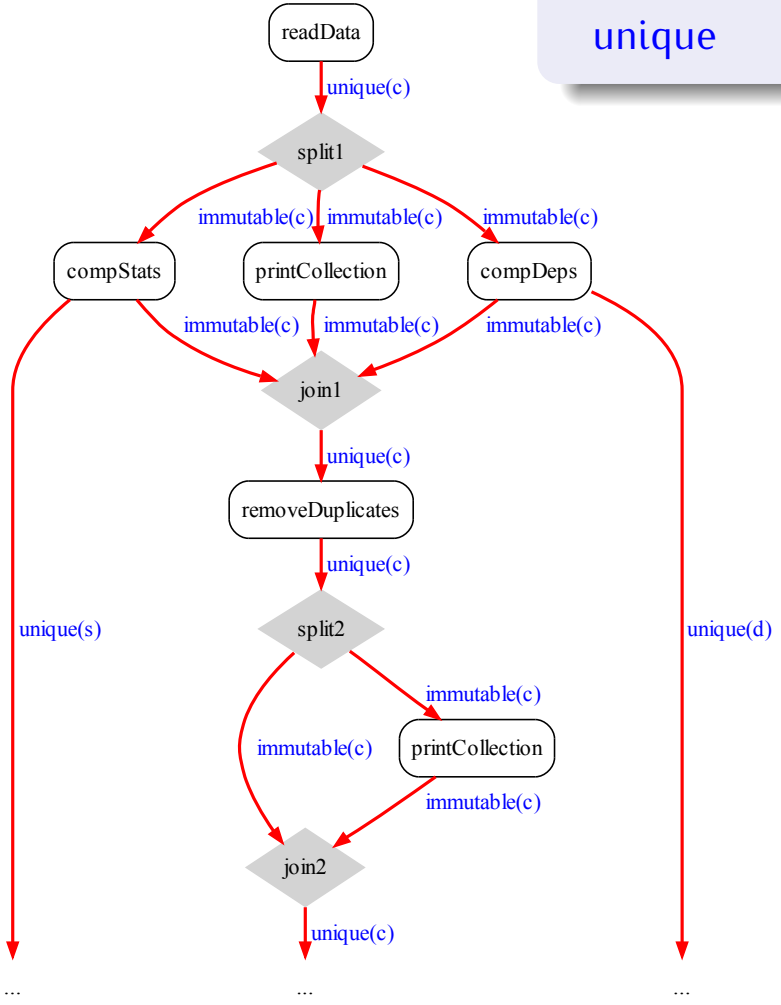
Statistics compStats(Collection c)
: **immutable**(c) ⇒ **immutable**(c), **unique**(**result**)

Solution

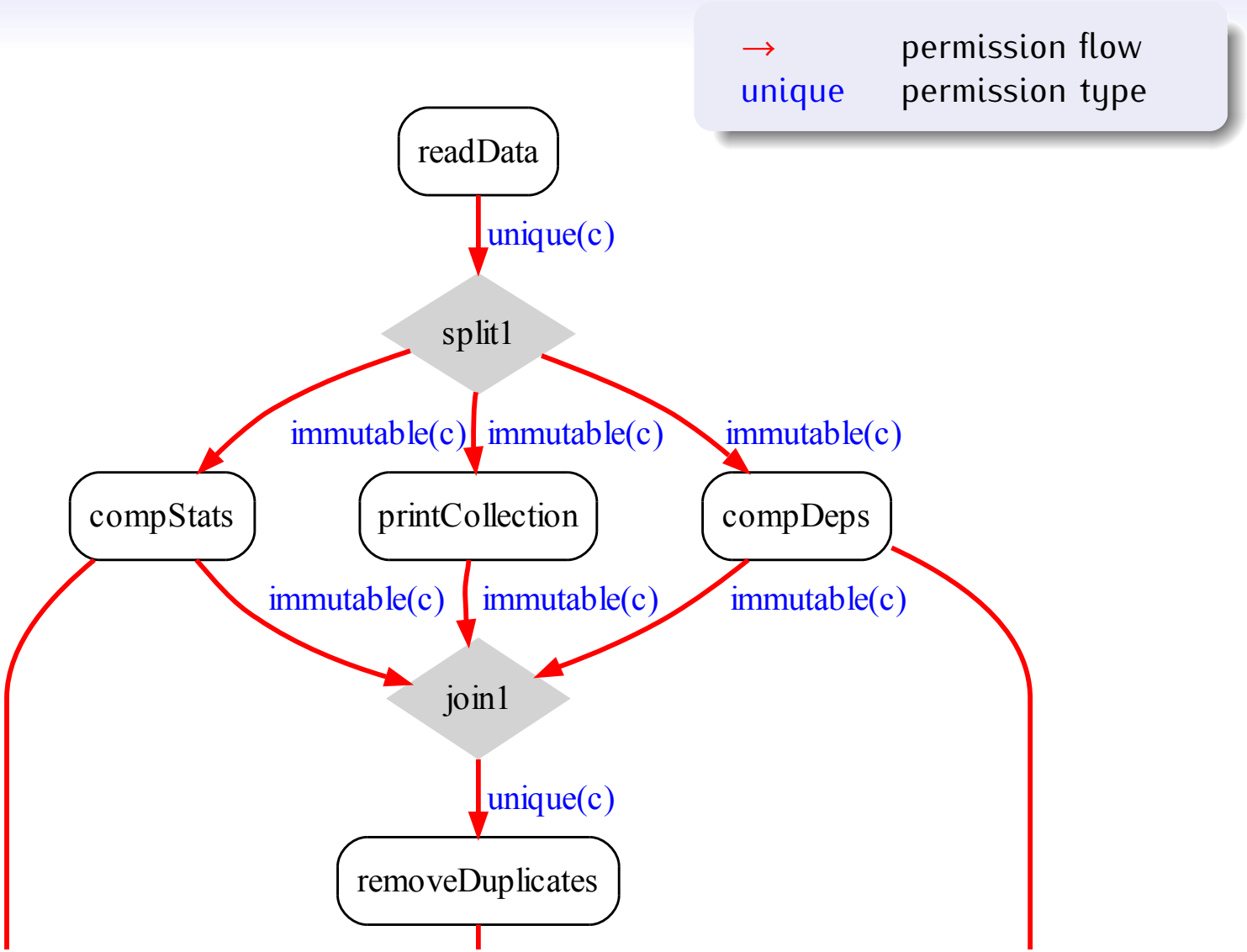
- permissions specify data access
- `compStats` and `compDeps` **do not** and **cannot** modify the collection
- permissions specification describes **contract**

Access Permissions

→	permission flow
unique	permission type



Access Permissions



Approach

Research Question

- What are the requirements for a runtime systems, to execute such programs efficiently ?
  - ⇝ How to use the **mixture fine - coarse grain concurrency information to execute a program efficiently**?

Possible Approaches

- using a **cost semantics** to estimate best granularity (static)
- system dependent **cost model** at start of program (semi-dynamic)
- **profiling** and **adaption** during **runtime** (dynamic)
- a bit of all above ...

Approach

Research Question

- How much concurrency can be extracted via such a high level abstraction ?
  - ⇝ How much **performance improvements** can **real world applications** achieve ?

Possible Approaches

- depends on granularity ... (see previous question)
- **build prototype** and **re-(write) real world applications** and **benchmark**

Standing on the Shoulders of Giants

Standing on the Shoulders of Giants

Access Permissions

Boyland	• Verification of concurrent programs (simple system with just unique/immutable) • Verification of correct lock usage (nested permissions)
Bierhoff	• Verification of API protocol conformance
Beckman	• Verification of correct usage of atomic blocks • Optimizations of Software Transactional Memory system
Terauchi	• Verification of correct usage of locks

Standing on the Shoulders of Giants

Concurrent Programming Languages

Fortress	• concurrent by default semantics for loops and tuples • no check for correct synchronization
Axum	• programs as dataflow graph • manual generation of graph
ML,etc	• exact dependency information • no state
NESL,etc.	• implicit concurrency model • largely limited to domain specific areas

Road Ahead

Road Ahead

Current Status

- worked on AtomicPower!
  - use access permissions to optimize STM
- developed core concepts of ÆMINIUM
  - **concurrency by default**
  - **access permissions**
  - **data groups**
- developed core language grammar
- published a position paper in Onward! conference

Road Ahead

Next Steps

Core Language (2nd - 3rd year)

- develop core language calculus
- proof core language calculus

Runtime System (3nd - 4th year)

- first step develop interpreter to gather experience
- implement efficient runtime system
- alternatively find existing runtime system
  - Axum, Dryad, ...

Evaluation (4th - 5th year)

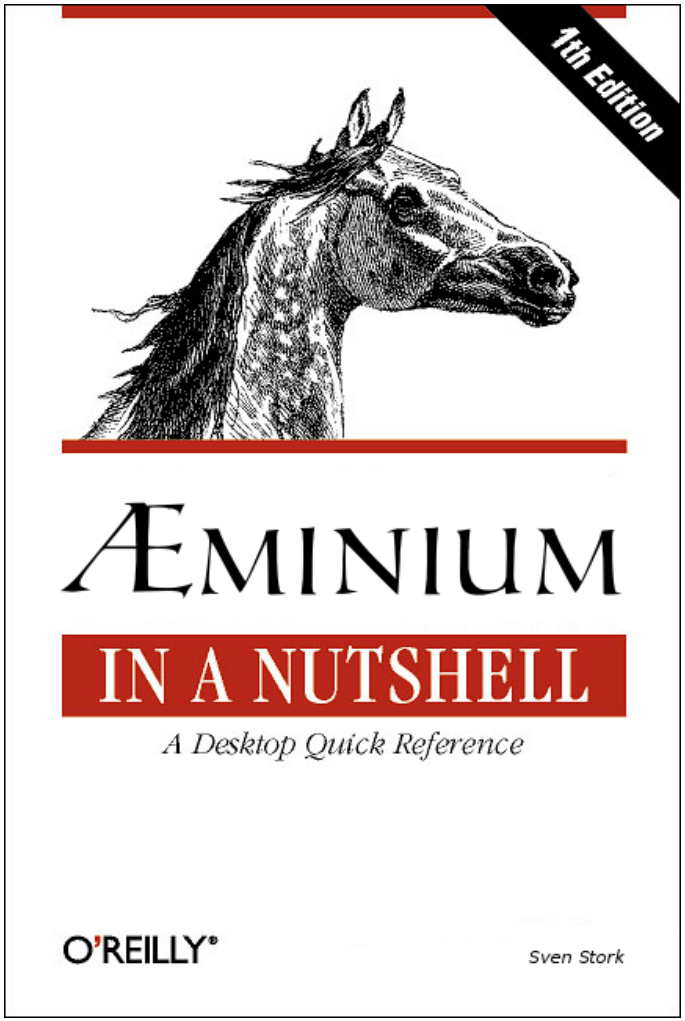
- evaluate performance via application benchmarks
- user studies to evaluate usability and 'real world code'

Road Ahead

Impact ... What if we achieve Nirvana ...

- Develop an programming languages (approach) that helps to write **less error prone concurrent code**.
- Educate or change mind set of programmers:
  - Permission **require some rigor** when designing interfaces
    $\rightsquigarrow$ **better design**
  - **Reasoning about dependencies instead of execution orders**
- An innovative runtime system, that **executes programs as efficient as dependencies and hardware permits**.

Summary


1th Edition
ÆMINIUM
IN A NUTSHELL
A Desktop Quick Reference
O'REILLY®
Sven Stork

Summary

Summary

- Proposed a new programming paradigm : Concurrency by Default
  - use **access permissions** and **data groups** to specify **dependencies** and extract **concurrency information**
- ÆMINIUM a new programming language based on concurrency by default
  - identified core features
  - developed grammar
- Future Work:
  - develop and proof formal system of ÆMINIUM
  - develop an efficient runtime system for ÆMINIUM
  - evaluation of the system and the overall approach

Thanks for the attention!
Questions ?