IBM Research

A Type System for Data-Centric Synchronization

Frank Tip and Jan Vitek†
Mandana Vaziri, Julian Dolby, Christian Hammer†

IBM T.J. Watson Research Center and
† Purdue University

Concurrency control is difficult

- Low-level data race occur when threads access a location concurrently (at least one write) with no synchronization
- Locking discipline involves non-local reasoning
- Even if every shared access is protected, data may still end up in an inconsistent state

A High-Level Data Race

```
class Vector  {
  Object[] data;
  int count;
  …
  synchronized int size() { … }

  synchronized boolean addAll(Collection c){
    int size = c.size();
    …
    while (it.hasNext())
        data[count++] = it.next();
     …
  }
```


synchronization is about preserving data consistency

so why not associate synchronization constraints directly with data?

Data-centric Synchronization with AJ

Data-Centric Synchronization: Terminology

```
class BankAccount {
   int checking, savings;
   int transferCount;

   void transfer(int amount) {
     synchronized(this) {
       checking -= amount;
       savings += amount;
     }
     transferCount++;
     …
   }

}
```


Data-Centric Synchronization: Terminology

```
class BankAccount {
   int checking, savings;
   int transferCount;

   void transfer(int amount) {
     synchronized(this) {
       checking -= amount;
       savings += amount;
     }
     transferCount++;
     ...
   }

}
```

atomic set S

Data-Centric Synchronization: Terminology

```
class BankAccount {
   int checking, savings;
   int transferCount;

   void transfer(int amount) {
      synchronized(this) {
         checking -= amount;
         savings += amount;
      }
      transferCount++;
      …
   }

}
```

atomic set S

unit of work on S

preserves consistency of S when executed sequentially

Atomic Sets

```
class Counter {
  atomicset a;
  atomic(a) int val;

  Counter(){ val = 0; }

  int get(){ return val; }
  void dec(){ val--; }
  void inc(){ val++; }
}
```

Atomic Sets

```
class Counter {
  atomicset a;
  atomic(a) int val;

  Counter(){ val = 0; }

  int get(){ return val; }
  void dec(){ val--; }
  void inc(){ val++; }
}
```

```
Counter c = new Counter();
```

Thread 1

```
c.inc();
c.dec();
```

Thread 2

```
c.inc();
```

`c.val` *has value 1 when both threads have terminated*

Aliasing

the atomic set b in the object pointed to by `this` *is merged with atomic set a in the Counter object*

```
class PairCounter {
  atomicset b;
  atomic(b) int diff;
  atomic(b) Counter|a=this.b| low  = new Counter|a=this.b|();
  atomic(b) Counter|a=this.b| high = new Counter|a=this.b|();

  void incHigh(){
    high.inc();
    diff = high.get()-low.get();
  }
}
```

unitfor

```
class Transfer {

  void transfer(unitfor(a) Counter from,
                unitfor(a) Counter to){
    from.dec();
    to.inc();
  }
}
```


A more realistic example

```
public abstract class AbsList {
  atomicset a;
  atomic(a) int size;

  public int size(){ return size; }
  public abstract ListIterator iterator();
  public abstract void add(Object o);
  public abstract boolean addAll(unitfor(a) AbsList c);
  …
}
```

A more realistic example (continued)

```
class LinkedList extends AbsList {
  atomic(a) Entry|b=this.a| header;

  public LinkedList(){
    header = new Entry|b=this.a|(null,null,null);
    header.next = header.prev = header;
  }

  …
}

internal class Entry {
  atomicset b;
  atomic(b) Object elem;
  atomic(b) Entry|b=this.b| next;
  atomic(b) Entry|b=this.b| prev;
  …
}
```

Implementing AJ

Implementation

- source-to-source translator with Eclipse
  - atomic set annotations entered as Java comments
  - implementation handles Java subset (no generics, inner classes
  - support alias annotations for arrays `|this.M[]F=this.M|`
  - translation generates standard synchronized block

Java Collections Framework

- Selected classes from Java Collections
  - ArrayList, LinkedList, HashMap, HashSet, TreeMap…+ java.util dependencies
  - 63 types and 10,860 LOC
- Introduced atomic sets
  - one atomicset for each of 5 subhierarchies, includes all instance fields
  - alias annotations to relate iterators to their “owner”
  - one class made internal (LinkedList.Entry)

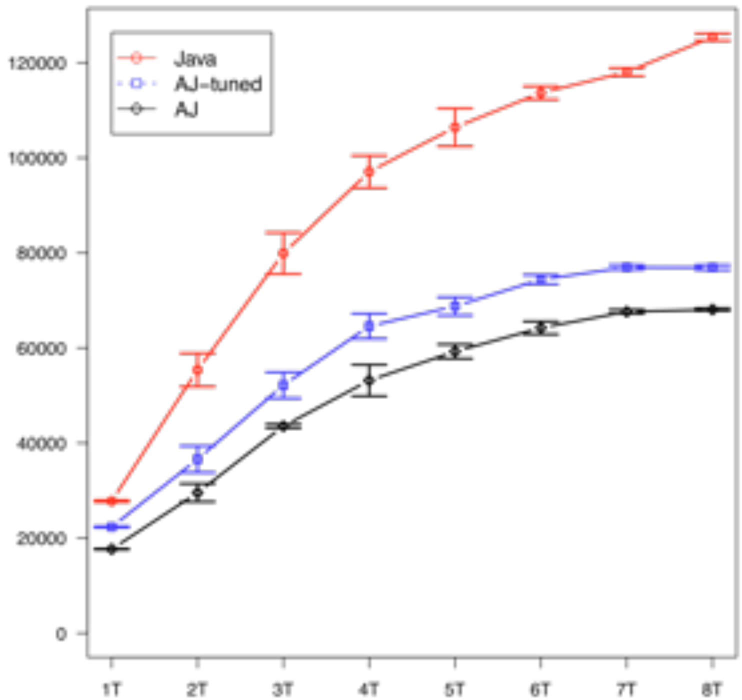
annotation	#
atomicset	**0**
atomic class	**5**
atomic	**0**
unitfor	**55**
alias	**330**
array object	**24**
array element	**16**
TOTAL	**430**

- ~1 annotation / 25 LOC

SPECjbb2005

- Widely used multi-threaded benchmark (8KLOC)
  - inconsistent/redundant synchronization
- Translation into AJ
  - for tuned version refactored some fields to make them final

type	#
atomicset	**1**
atomic class	**14**
atomic	**25**
unitfor	**0**
alias	**8**
array object	**0**
array element	**1**
TOTAL	**49**



- ~1 annotation / 160 LOC
- removed 125 occurrences of synchronized

25 synchronized remain, related to wait/notify

Formalizing AJ

Atomic Set Serializablity

- AJ guarantees

Theorem

Given a well-formed trace T *and atomic set* R,
events of each units of work of R *happen serially.*

Modeling AJ

To be tractable, select a subset of AJ:

(FeatherweightJava + state + atomicsets) + threads

or

AJ - *unitfor - arrays - modifiers - generics - primitives - exceptions - finals - statics - new_threads - interfaces*

$$
\begin{array}{rcll}
p & ::= & \overline{cd} & \textit{program} \\
cd & ::= & \iota\ \textsf{class C extends D}\ \{\ as\ \overline{fd}\ \overline{md}\ \} & \textit{class} \\
as & ::= & \textsf{atomicset a} \mid \epsilon & \\
fd & ::= & \alpha\ \tau\ \textsf{f} & \textit{field} \\
md & ::= & \tau\ \textsf{m}\ (\overline{\tau\ \textsf{x}})\ \{\ \overline{\tau\ \textsf{z}};\textsf{s};\textsf{return y}\ \} & \textit{method} \\
\textsf{s} & ::= & \textsf{s;s} \mid \textsf{skip} \mid \textsf{x =this.f} \mid \textsf{x =}(\tau)\textsf{y} \mid & \textit{statement} \\
 & & \textsf{this.f =z} \mid \textsf{x =new}\ \tau\ () \mid \textsf{x =y.m}\ (\overline{\textsf{z}}) &
\end{array}
$$

$$
\begin{array}{rcll}
\tau & ::= & \textsf{C}|\textsf{a}=\textsf{this.b}| \mid \textsf{C} & \textit{type} \\
\alpha & ::= & \textsf{atomic (a)} \mid \epsilon & \\
\iota & ::= & \textsf{internal} \mid \epsilon & \\
E & ::= & [] \mid E[x : \tau] & \textit{type env}
\end{array}
$$

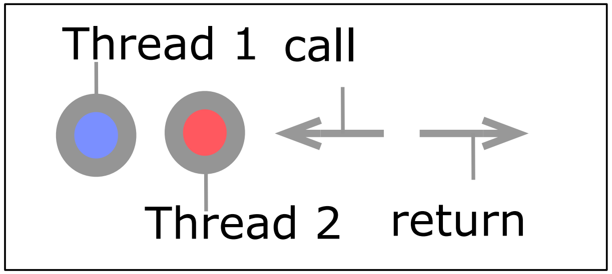
Abstracting Java executions

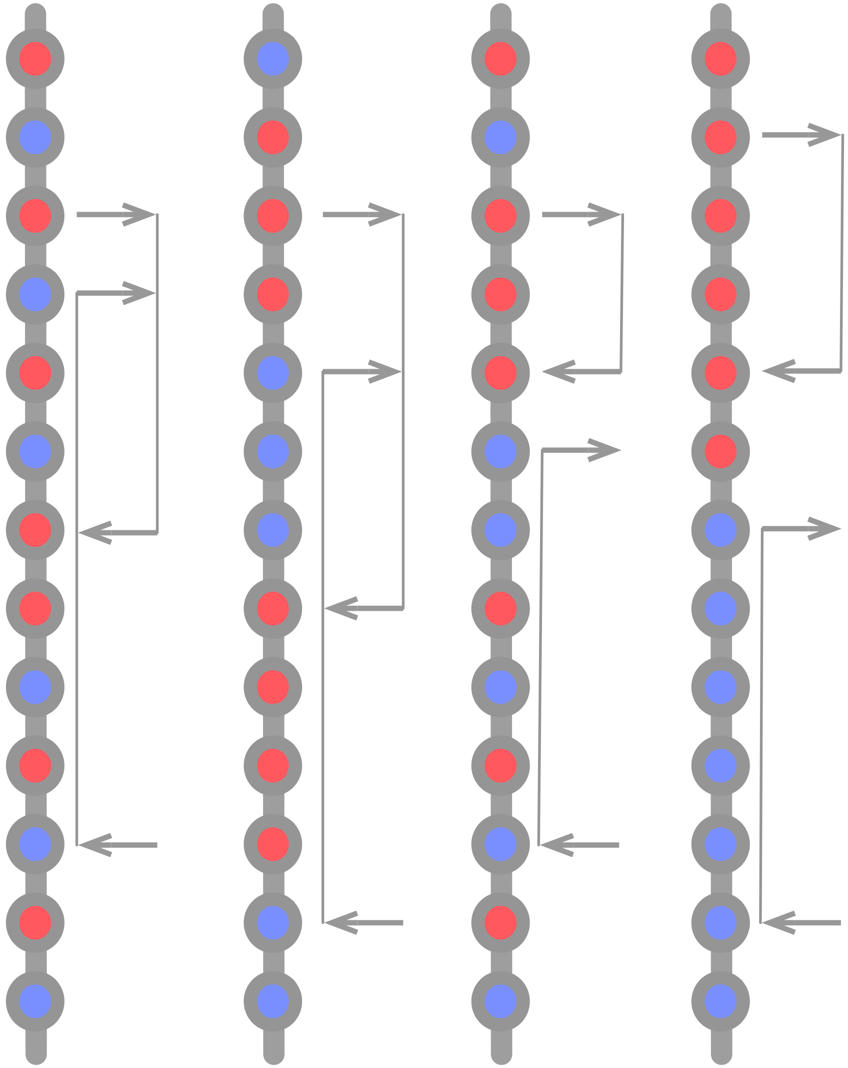
- Dynamic Semantics

$$H; \overline{T}\, T\, \overline{T'} \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'$$

- Scheduling is non-deterministic
- Traces are sequences of events:

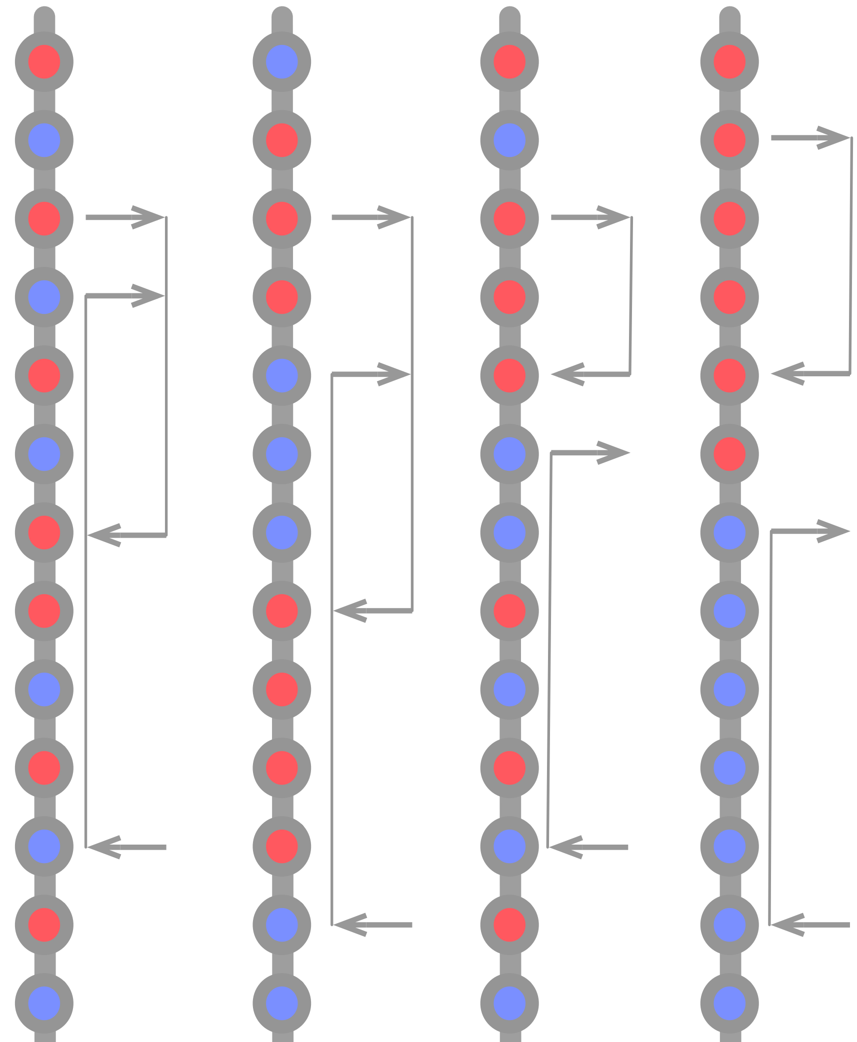
  read x.f, write x.f, call x.m, return





Modeling AJ executions

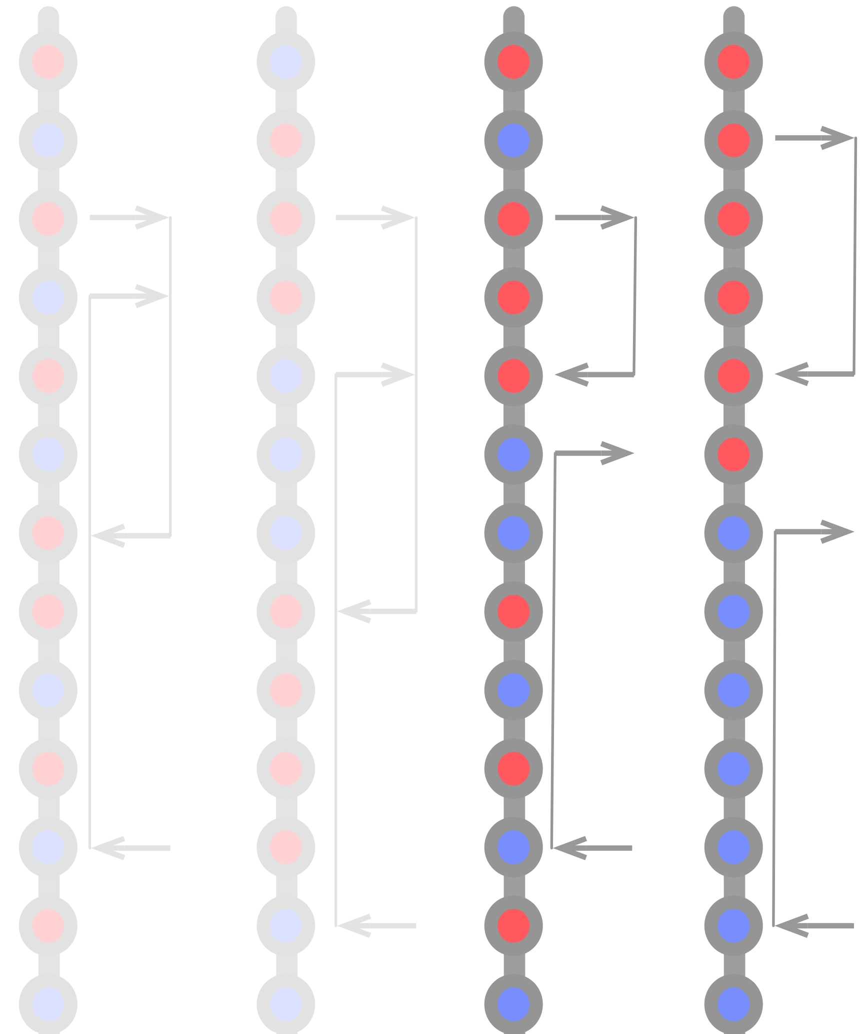
- Units of work on same atomicset are mutually exclusive
- Modeled by restricting valid traces



Definition *An event* $e = (H, \overline{T}, \ell, \rho)$ *is* valid *if and only if, when* $\ell = \longrightarrow r.\mathsf{m}$, $H(r) = \mathsf{C}|r'|(\overline{r})$ *and* C not *internal then* $\nexists\, \rho' S \in \overline{T}.\rho' \neq \rho$ *and* $\langle \mathsf{m}\, F\ \ \mathsf{s} \rangle \in S$ *and* $H(F(\mathsf{this})) = \mathsf{D}|r'|(\overline{\mathsf{z}})$.

Modeling AJ executions

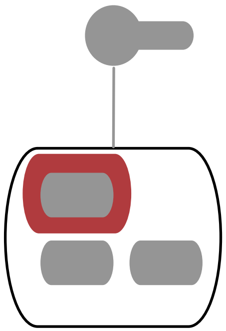
- Units of work on same atomicset are mutually exclusive
- Modeled by restricting valid traces



Definition *An event* $e = (H, \overline{T}, \ell, \rho)$ *is* valid *if and only if, when* $\ell = \longrightarrow r.\mathsf{m}$, $H(r) = \mathsf{C}|r'|(\overline{r})$ *and* C *not* internal *then* $\nexists\, \rho' S \in \overline{T}.\rho' \neq \rho$ *and* $\langle \mathsf{m}\, F \;\; \mathsf{s}\rangle \in S$ *and* $H(F(\mathsf{this})) = \mathsf{D}|r'|(\overline{\mathsf{z}})$.

What could go wrong?

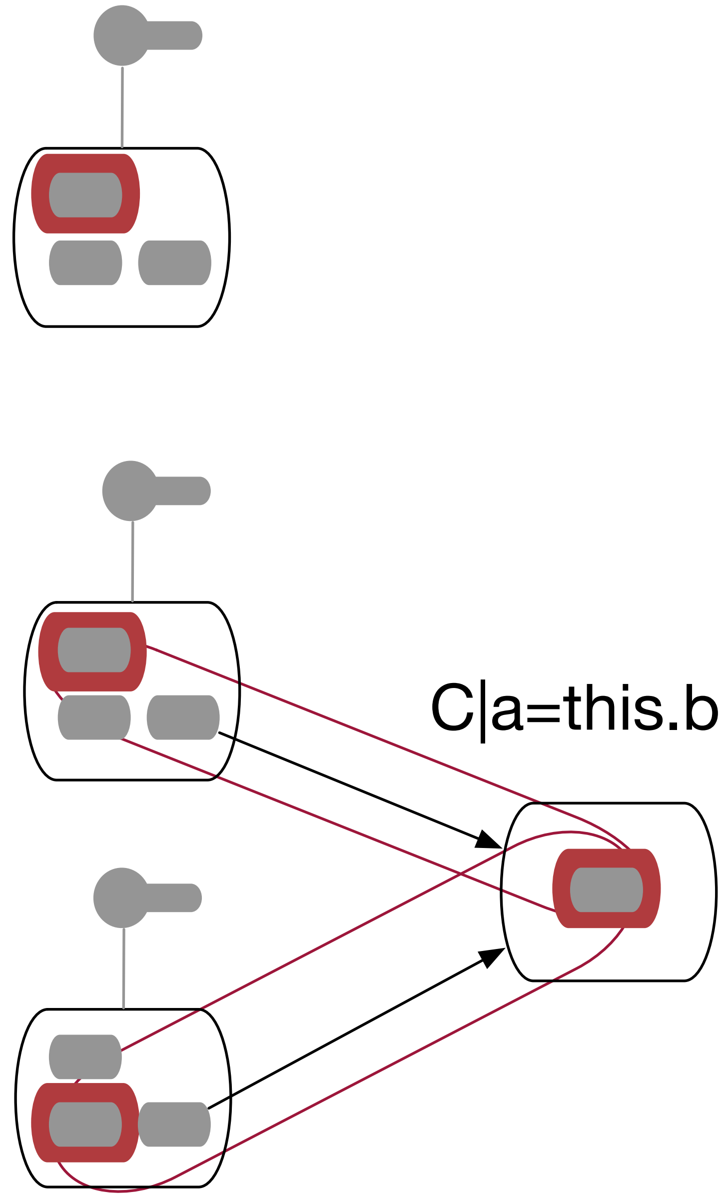
What could go wrong?



- *A field is accessed outside of the boundaries of a unit of work*

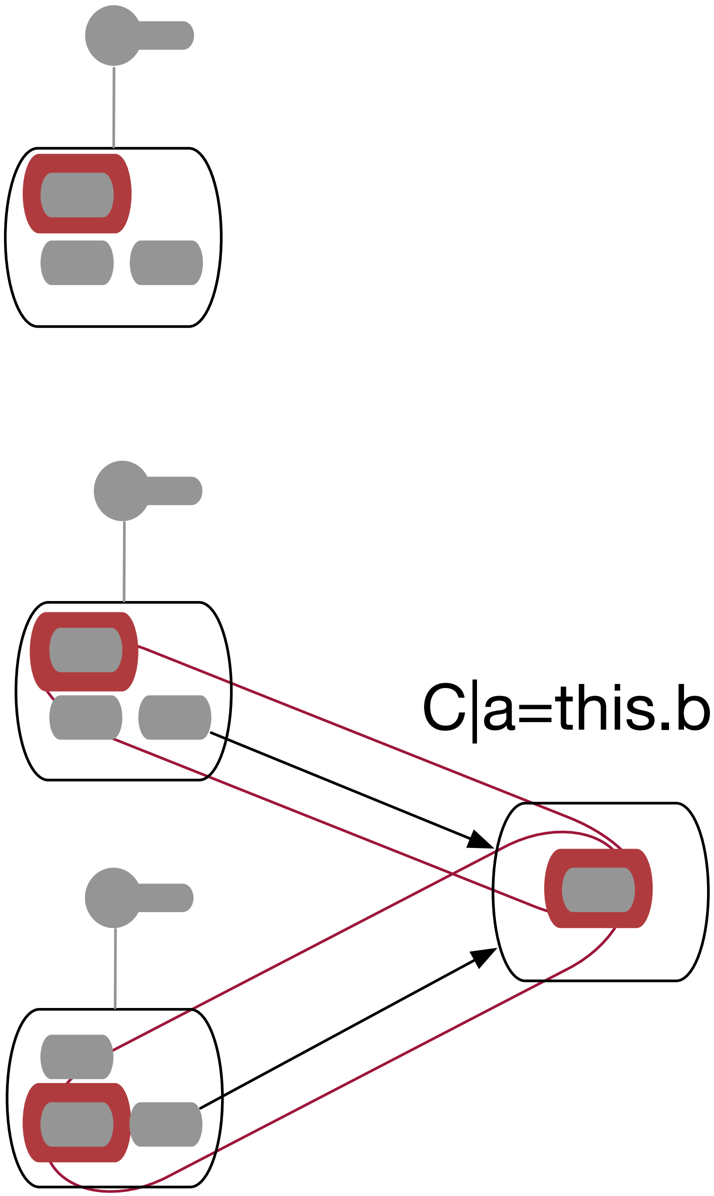
What could go wrong?

- *A field is accessed outside of the boundaries of a unit of work*
- Could this happen?
  – Alias confusion… an object is referenced from two atomicsets



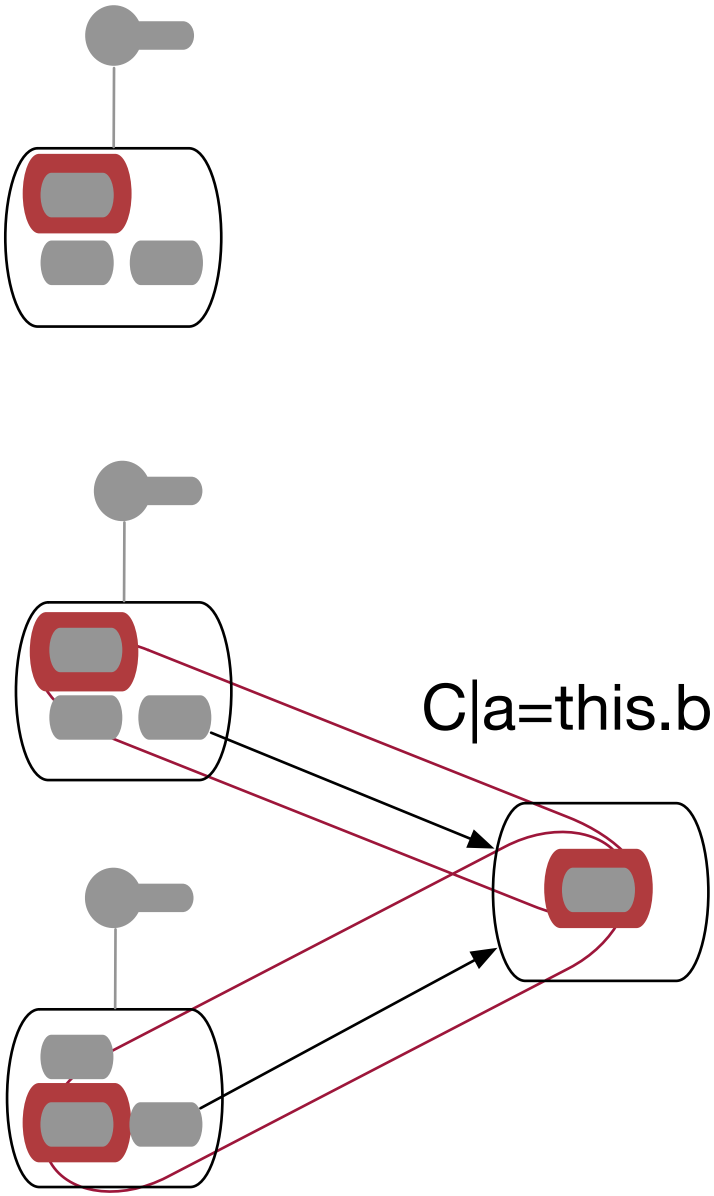
What could go wrong?

- *A field is accessed outside of the boundaries of a unit of work*
- Could this happen?
  - Alias confusion… an object is referenced from two atomicsets
  - Ownership leak… an object is directly referenced from outside of its atomicset



What could go wrong?

- *A field is accessed outside of the boundaries of a unit of work*
- Could this happen?
  - Alias confusion… an object is referenced from two atomicsets
  - Ownership leak… an object is directly referenced from outside of its atomicset
- A well-formed configuration is one, where this does not occur



Technically,

$$\frac{H \text{ is WF in } H \quad \overline{T} \text{ is WF in } H \quad \vdash CT}{H; \overline{T} \text{ is WF}}\ (\text{WF-CONFIGURATION})$$

$$\frac{}{[]\ \text{ is WF in } H}\ (\text{WF-EMPTY-HEAP})$$

$$\frac{}{\rho\, \mathsf{NPE} \text{ is WF in } H}\ (\text{WF-NPE-THREAD})$$

$$\frac{\langle \mathsf{run}\ F\ \ \mathsf{s}\rangle \text{ is WF in } H \qquad \mathit{not\ internal}_H(F(\mathsf{this}))}{\rho\,\langle \mathsf{run}\ F\ \ \mathsf{s}\rangle \text{ is WF in } H}\ (\text{WF-THREAD-BOT})$$

$$\frac{\begin{array}{c}\langle \mathsf{m}\ F\ \ \mathsf{s}\rangle \text{ is WF in } H \quad S \equiv S'\langle \mathsf{m}'\ F'\ \ \mathsf{x} = \mathsf{y}.\mathsf{m}(\overline{\mathsf{z}'}); \mathsf{s}''\rangle \\ \rho\, S \text{ is WF in } H \\ (\exists \langle \mathsf{m}''\ F''\ \ s''\rangle \in S\langle \mathsf{m}\ F\ \ \mathsf{s}\rangle,\ \mathit{not\ internal}_H(F''(\mathsf{this})) \\ \mathit{and\ owner}_H(F''(\mathsf{this})) = \mathit{owner}_H(F(\mathsf{this})))\end{array}}{\rho\, S\langle \mathsf{m}\ F\ \ \mathsf{s}\rangle \text{ is WF in } H}\ (\text{WF-THREAD})$$

$$\frac{\begin{array}{c}(\mathsf{C}\ \mathit{has}\ \mathsf{a}\ \mathit{implies}\ \omega \neq \epsilon) \quad H' \text{ is WF in } H \\ \mathit{fields}(\mathsf{C}) = \overline{\alpha\,\tau\,\mathsf{f}} \quad \overline{r\mathsf{z} <:_{r,H} \tau}\end{array}}{H'[r \mapsto \mathsf{C}|\omega|(\overline{r\mathsf{z}})]\ \text{ is WF in } H}\ (\text{WF-HEAP})$$

$$\frac{\begin{array}{c}\mathit{locals}(\mathsf{m}, F) = E \quad E \vdash \mathsf{s} \\ \forall\, \mathsf{x} \in \mathit{dom}(F), F(\mathsf{x}) <:_{F(\mathsf{this}),H} E(\mathsf{x})\end{array}}{\langle \mathsf{m}\ F\ \ \mathsf{s}\rangle \text{ is WF in } H}\ (\text{WF-FRAME})$$

Actually,

- Well-formedness ensures that
  - all threads are in a consistent state
  - heap is well-typed & no aliasing confusion & no ownership leaks

- We prove that AJ programs preserve WF properties

Theorem 1. *Preservation. If $H; \overline{T}\, T\, \overline{T'}$ is WF and $H; \overline{T}\, T\, \overline{T'} \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'$, then $H; \overline{T}\, \overline{T'}\, T'$ is WF.*

Theorem 2. *Progress. If $H; \overline{T}\, T\, \overline{T'}$ is WF and $active(T)$, then $H; \overline{T}\, T\, \overline{T'} \xrightarrow{\ell}_{\rho} H'; \overline{T}\, \overline{T'}\, T'$.*

AJ Type System

- subtyping rules
- viewpoint adaption
- type rule for classes, methods, and statement

Subtyping

- Simple transitive, reflexive closure of the extends relation
- Alias types are define a distinct relation
- The language requires explicit casts for type changes

$$\frac{}{\mathsf{C} <: \mathsf{C}} \qquad \frac{\mathsf{C}\ \text{extends}\ \mathsf{D}}{\mathsf{C} <: \mathsf{D}} \qquad \frac{\mathsf{C} <: \mathsf{C}' \quad \mathsf{C}' <: \mathsf{D}}{\mathsf{C} <: \mathsf{D}}$$

$$\frac{\mathsf{C} <: \mathsf{D}}{\mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| <: \mathsf{D}|\mathsf{a}{=}\mathsf{this.b}|}$$

Typing a Class

- The internal annotation of parents must be preserved
- (single atomic set restriction is enforced)
- Must re-check inherited methods

$$\frac{\overline{fd}\ \mathrm{OK\ in}\ \mathsf{C} \quad methods(\mathsf{C}) = \overline{md'} \quad \overline{md'}\ \mathrm{OK\ in}\ \mathsf{C} \quad (\mathsf{D}\ has\ \mathsf{a}\ \ implies\ \ as = \epsilon) \quad (\iota = \mathsf{internal}\ \ implies\ \ \mathsf{C}\ has\ \mathsf{a}) \quad (\mathsf{D}\ is\ \mathsf{internal}\ implies\ \iota = \mathsf{internal})}{\iota\ \mathsf{class\ C\ extends\ D}\ \{\ as\ \overline{fd}\ \overline{md}\ \}\ \mathrm{OK}}$$

Typing a Class

- The internal annotation of parents must be preserved
- (single atomic set restriction is enforced)
- Must re-check inherited methods

$$\frac{\overline{fd}\ \text{OK in C} \quad methods(\mathsf{C}) = \overline{md'} \quad \overline{md'}\ \text{OK in C} \quad (\mathsf{D}\ has\ \mathsf{a}\ implies\ as = \epsilon) \quad \boxed{(\iota = \mathsf{internal}\ implies\ \mathsf{C}\ has\ \mathsf{a}) \quad (\mathsf{D}\ is\ \mathsf{internal}\ implies\ \iota = \mathsf{internal})}}{\iota\ \mathsf{class\ C\ extends\ D}\ \{\ as\ \overline{fd}\ \overline{md}\ \}\ \text{OK}}$$

Typing a Class

- The internal annotation of parents must be preserved
- (single atomic set restriction is enforced)
- Must re-check inherited methods

$$\frac{\overline{fd} \text{ OK in C} \quad methods(\mathsf{C}) = \overline{md'} \quad \overline{md'} \text{ OK in C} \quad (\mathsf{D}\ has\ \mathsf{a}\ \ implies\ \ as = \epsilon) \quad \boxed{(\iota = \mathsf{internal}\ \ implies\ \ \mathsf{C}\ has\ \mathsf{a}) \quad (\mathsf{D}\ is\ \mathsf{internal}\ implies\ \iota = \mathsf{internal})}}{\iota\ \mathsf{class\ C\ extends\ D}\ \{\ as\ \overline{fd}\ \overline{md}\ \}\ \text{OK}}$$

Typing statements

- Casting off an alias type is allowed for non-internal classes
- Casting between alias types must preserve aliases

$$\frac{E(\mathsf{x}) = \mathsf{D}|\mathsf{a}{=}\mathsf{this.b}| \quad E(\mathsf{y}) = \mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| \qquad \mathsf{C}\ \mathit{has}\ \mathsf{a} \quad E(\mathsf{this})\ \mathit{has}\ \mathsf{b} \quad \boxed{\mathsf{D} <: \mathsf{C}}}{E \vdash \mathsf{y} = \boxed{(\mathsf{C}|\mathsf{a}{=}\mathsf{this.b}|)\mathsf{x}}}$$

$$\frac{\boxed{E(\mathsf{x}) = \mathsf{C}|\mathsf{a}{=}\mathsf{this.b}| \quad \mathsf{C}\ \mathit{not}\ \mathsf{internal}} \qquad E(\mathsf{y}) = \mathsf{C}}{E \vdash \mathsf{y} = \boxed{(\mathsf{C})\mathsf{x}}}$$

Typing statements

- Method calls require viewpoint adaption of arguments & return type

$$adapt(\mathsf{C}, \tau) \;=\; \mathsf{C}$$

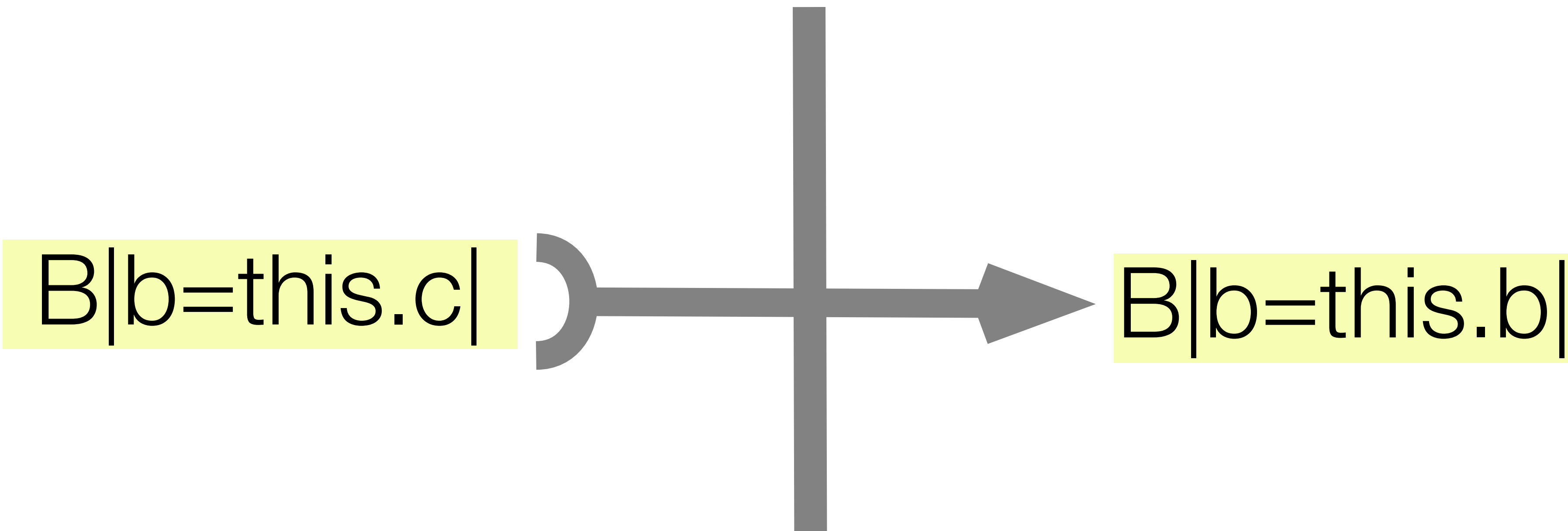
$$adapt(\mathsf{C}|\mathsf{a}{=}\mathsf{this.b}|, \mathsf{D}|\mathsf{b}{=}\mathsf{this.c}|) \;=\; \mathsf{C}|\mathsf{a}{=}\mathsf{this.c}|$$

$$\frac{E(\mathsf{y}) = \tau_{\mathsf{y}} \quad typeof(\tau_{\mathsf{y}}.\mathsf{m}) = \overline{\tau} \to \tau \quad E(\overline{\mathsf{z}}) = \overline{\tau_{\mathsf{z}}} \qquad \overline{\tau_{\mathsf{z}}} = adapt(\overline{\tau}, \tau_{\mathsf{y}}) \quad \tau' = adapt(\tau, \tau_{\mathsf{y}}) \quad E(\mathsf{x}) = \tau'}{E \vdash \mathsf{x} = \mathsf{y}.\mathsf{m}(\overline{\mathsf{z}})}$$

View point adaption

```
class C { atomicset c;
  B|b=this.c| x;
  x=new B|b=this.c|();
  x.gee(x)
```

```
class B { atomicset b;
 void gee(B|b=this.b| x){
```



Related Work

- *See the paper for a complete list*
- Atomic set
  - Vaziri e/a [POPL06] *original proposal*
  - *detection of atomicset serializability violations*
    - Hammer e/a [ICSE08] *dynamic*
    - Kidd e/a [VMCAI09] *static*
- Data groups
  - Leino e/a [OOPSLA98] *abstract rep. of groups of fields for modular reasoning*
- Atomicity and Race-free Types
  - Flanagan e/a [ESOP99,PLDI00,PLDI03,TOPLAS08, ∞]
- Ownership types
  - Zhao e/a [JFP06] *lightweight ownership disciplines*
- Lock inference
  - Cherem e/a. [PLDI08], McCloskey e/a [POPL06]

Conclusions

- A type system for data-centric synchronization which
  - guarantees atomic-set serializablility
  - enables separate compilation
  - handles unbounded sets

- Annotation overhead competitive with race-freedom and atomicity type systems

- Future
  - improve performance
  - local annotation inference
  - static deadlock prevention