Fast and precise static race detection for loop-parallelism in Java programs

Cosmin Rădoi (joint work with Danny Dig)

UPCRC Illinois
Universal Parallel Computing Research Center

Why not dynamic race detectors?

existing dynamic analyses...

... require running the program

=> usually have runtime overhead

=> input dependent

=> do not catch all races

Why not an existing static race detector?

existing static analyses...

... improved a lot in the last decade

but,

- still do not scale or

- report many false races

=> less used than dynamic analysis

Focus

Can better results be achieved by focusing on specific parallelism pattern, e.g. loop parallelism?

Yes.

This allows cheaper and more precise solutions for aliasing, lock identity and array sensitivity.

Why loop parallelism?

common practice for speeding up software

well supported by parallel frameworks

- C++ TBB: parallel_for
- .NET TPL: Parallel.For
- Java: ParallelArray

Why race detection?

non-determinism

=> concurrency bugs, including races, are hard to find

OO programs frequently share data through the heap

=> races are common

Motivating example

```
particles.generate(new Generator<Particle>() {
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});
for (int i = 0; i < noSteps; i++) {
    // ... update force ...

    particles.apply(new Procedure<Particle>() {
        public void op(Particle p) {
            move(p);
        }
    });
    particles.apply(new Procedure<Particle>() {
        public void op(Particle p) {
            computeCenterOfMass();
        }
    });
}
```

Motivating example

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});
for (int i = 0; i < noSteps; i++) {
    // ... update force ...

    particles.apply(...
        public void op(Particle p) {
            move(p)
        }
    });
    particles.apply(...
        public void op(Particle p) {
            computeCenterOfMass();
        }
    });
}
```

Motivating example

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});
for (int i = 0; i < noSteps; i++) {
    // ... update force ...

    particles.apply(...
        public void op(Particle p) {
            move(p)
        }
    });
    particles.apply(...
        public void op(Particle p) {
            computeCenterOfMass();
        }
    });
}
```

Motivating example

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});

[...]

particles.apply(...
    public void op(Particle p) {
        move(p)
    }
});
```

Motivating example

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});

[...]

particles.apply(...
    public void op(Particle p) {
        p.vX += p.fX / p.m * dT;
        p.vY += p.fY / p.m * dT;
        p.x += p.vX * dT;
        p.y += p.vY * dT;
    }
});
```

Previous approaches reason about one thread

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});

[...]

particles.apply(...
    public void op(Particle p) {
        p.vX += p.fX / p.m * dT;
        p.vY += p.fY / p.m * dT;
        p.x += p.vX * dT;
        p.y += p.vY * dT;
    }
});
```

other

T

Previous approaches reason about one thread

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});

[...]

particles.apply(...
    public void op(Particle p) {
        p.vX += p.fX / p.m * dT;
        p.vY += p.fY / p.m * dT;
        p.x += p.vX * dT;
        p.y += p.vY * dT;
    }
});
```

T

other

Our approach reasons about two threads

T_1

```
particles.generate(...
    public Particle op() {
        Particle p1 = new Particle();
        readParticle(p1);
        return p1;
    }
});

[...]

particles.apply(...
    public void op(Particle p1) {
        p1.vX += p1.fX / p1.m * dT;
        p1.vY += p1.fY / p1.m * dT;
        p1.x += p1.vX * dT;
        p1.y += p1.vY * dT;
    }
});
```

T_2

```
particles.generate(...
    public Particle op() {
        Particle p2 = new Particle();
        readParticle(p2);
        return p2;
    }
});

[...]

particles.apply(...
    public void op(Particle p2) {
        p2.vX += p2.fX / p2.m * dT;
        p2.vY += p2.fY / p2.m * dT;
        p2.x += p2.vX * dT;
        p2.y += p2.vY * dT;
    }
});
```

Our approach reasons about two threads

T_1

```
particles.generate(...
    public Particle op() {
        Particle p1 = new Particle();
        readParticle(p1);
        return p1;
    }
});

[...]

particles.apply(...
    public void op(Particle p1) {
        p1.vX += p1.fX / p1.m * dT;
        p1.vY += p1.fY / p1.m * dT;
        p1.x += p1.vX * dT;
        p1.y += p1.vY * dT;
    }
});
```

T_2

```
particles.generate(...
    public Particle op() {
        Particle p2 = new Particle();
        readParticle(p2);
        return p2;
    }
});

[...]

particles.apply(...
    public void op(Particle p2) {
        p2.vX += p2.fX / p2.m * dT;
        p2.vY += p2.fY / p2.m * dT;
        p2.x += p2.vX * dT;
        p2.y += p2.vY * dT;
    }
});
```

Motivating example

```
particles.generate(...
    public Particle op() {
        Particle p = new Particle();
        readParticle(p);
        return p;
    }
});
for (int i = 0; i < noSteps; i++) {
    // ... update force ...

    particles.apply(...
        public void op(Particle p) {
            move(p)
        }
    });
    particles.apply(...
        public void op(Particle p) {
            computeCenterOfMass();
        }
    });
}
```

Race when computing center of mass

O

```
... new Computation() ...

class Computation {
    // center of mass
    Particle c = new Particle();

    void compute() {
        [...]
```

T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

T_2

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

Race when computing center of mass

O

```
... new Computation() ...

class Computation {
    // center of mass
    Particle c = new Particle();

    void compute() {
        [...]
```

T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p1.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

T_2

```
particles.apply(...
    public void op(Particle p2) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p2.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

Race when computing center of mass

T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p1.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

T_2

```
particles.apply(...
    public void op(Particle p2) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p2.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

Race when computing center of mass

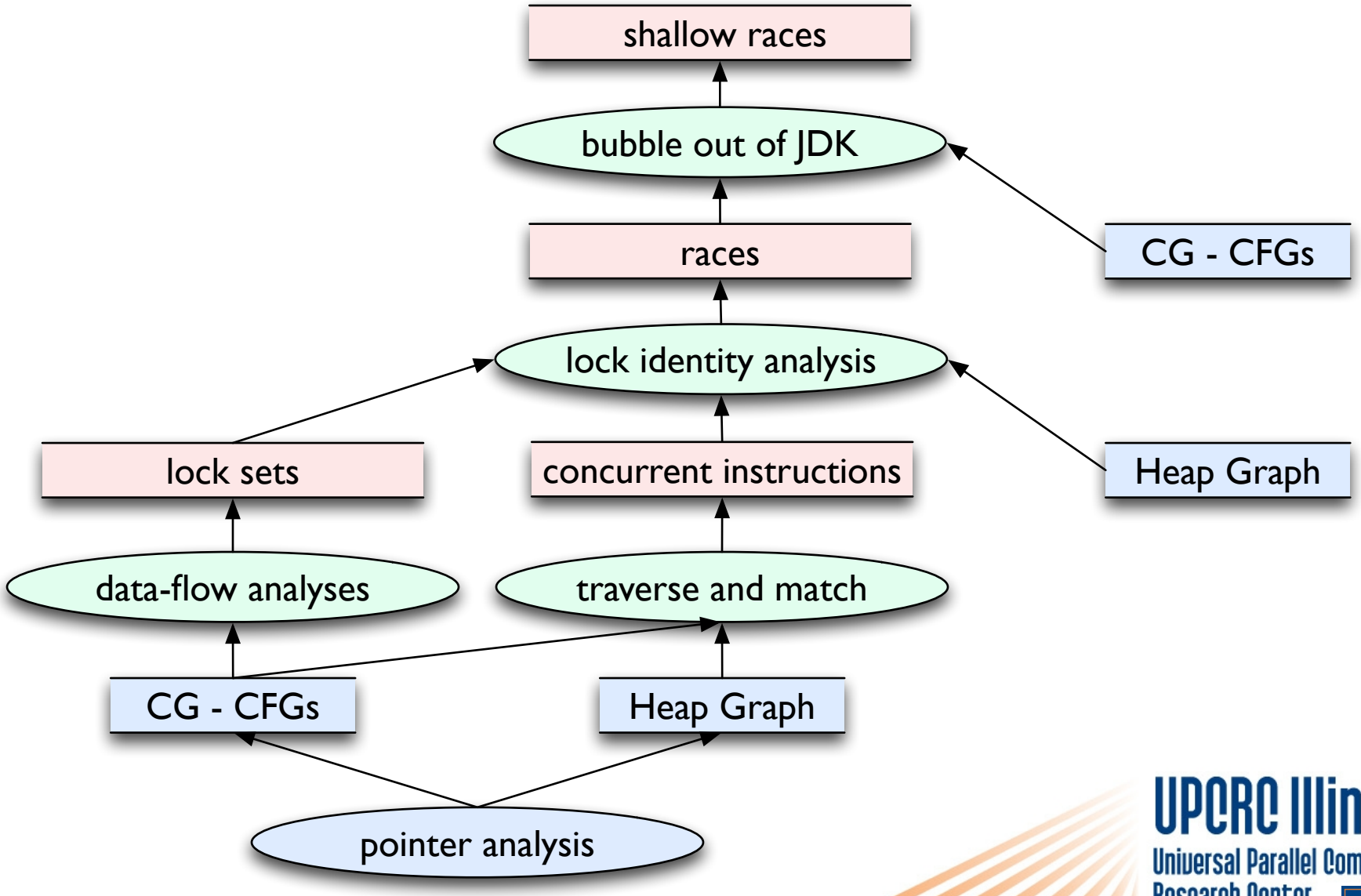
T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p1.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

T_2

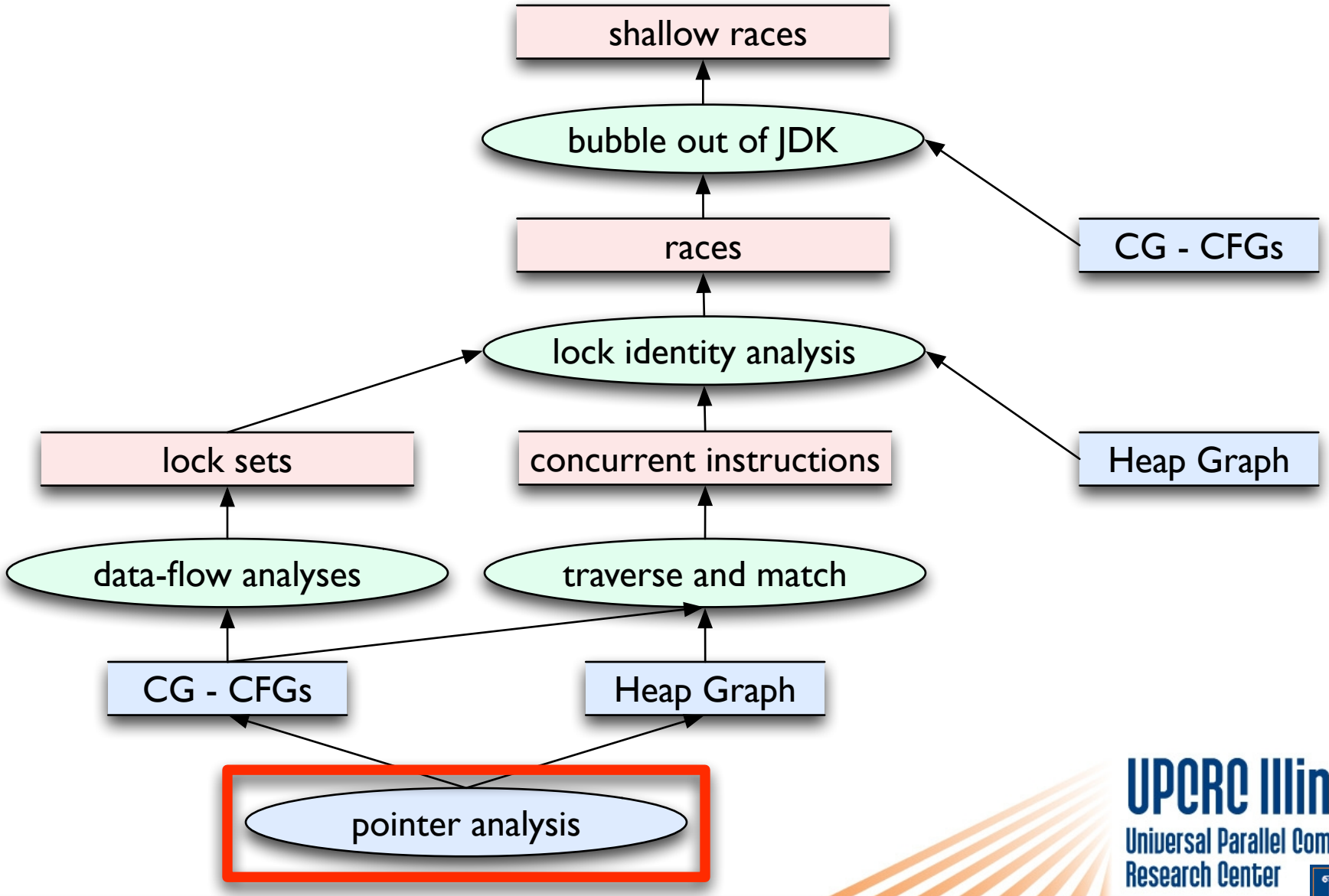
```
particles.apply(...
    public void op(Particle p2) {
        Particle oldC = c;
        this.c = new Particle();
        c.m = oldC.m + p2.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

System overview



UPCRC Illinois
Universal Parallel Computing Research Center

System overview



UPCRC Illinois
Universal Parallel Computing Research Center

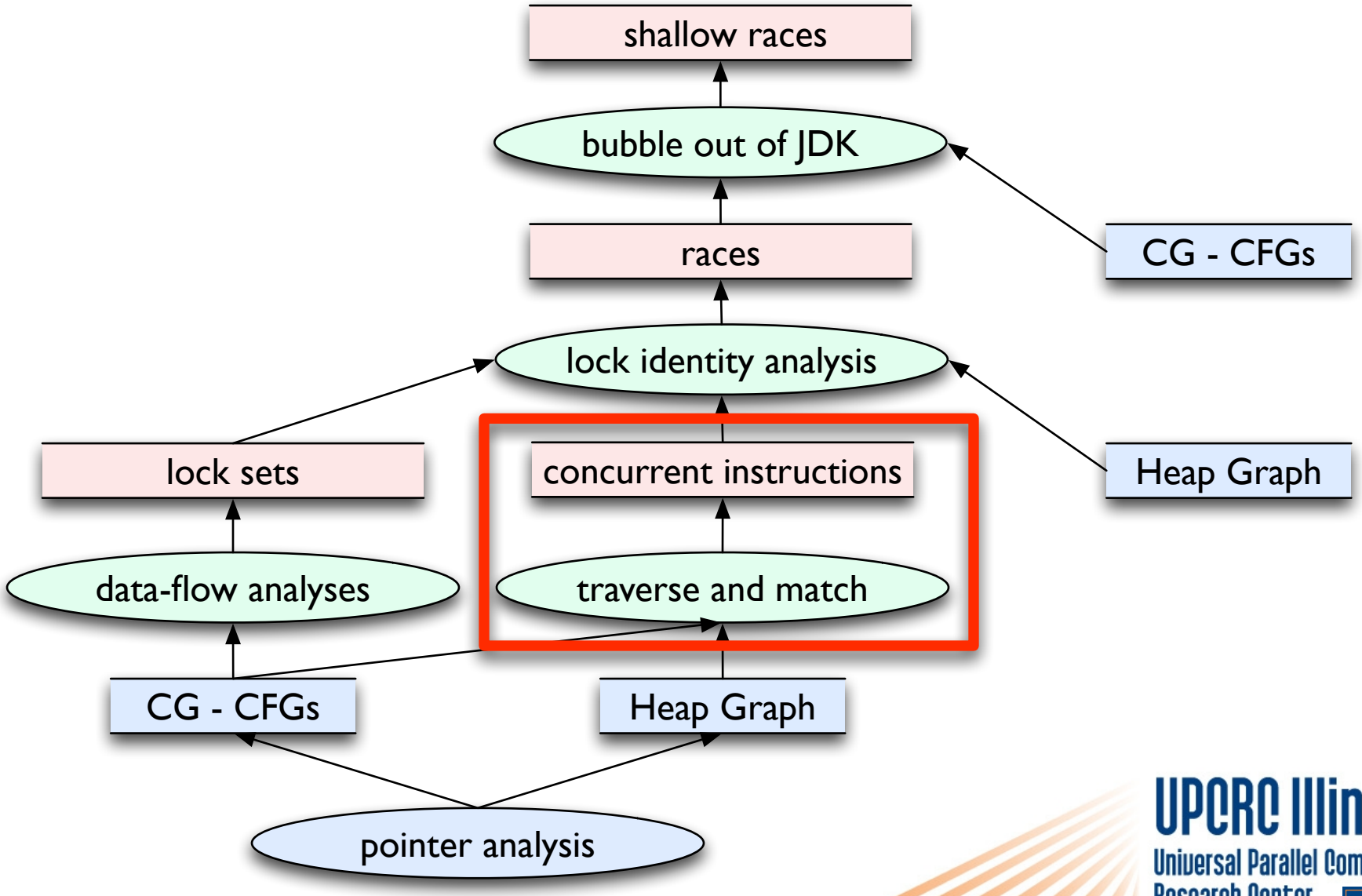
Pointer analysis

Andersen style analysis

special context sensitivity:

- 0-CFA to 1-CFA outside of loops, depending on the number of arrays the tool reasons about
- array object and loop iteration (our modeling)
- 1-object sensitivity on edge of JDK

System overview



UPCRC Illinois
Universal Parallel Computing Research Center

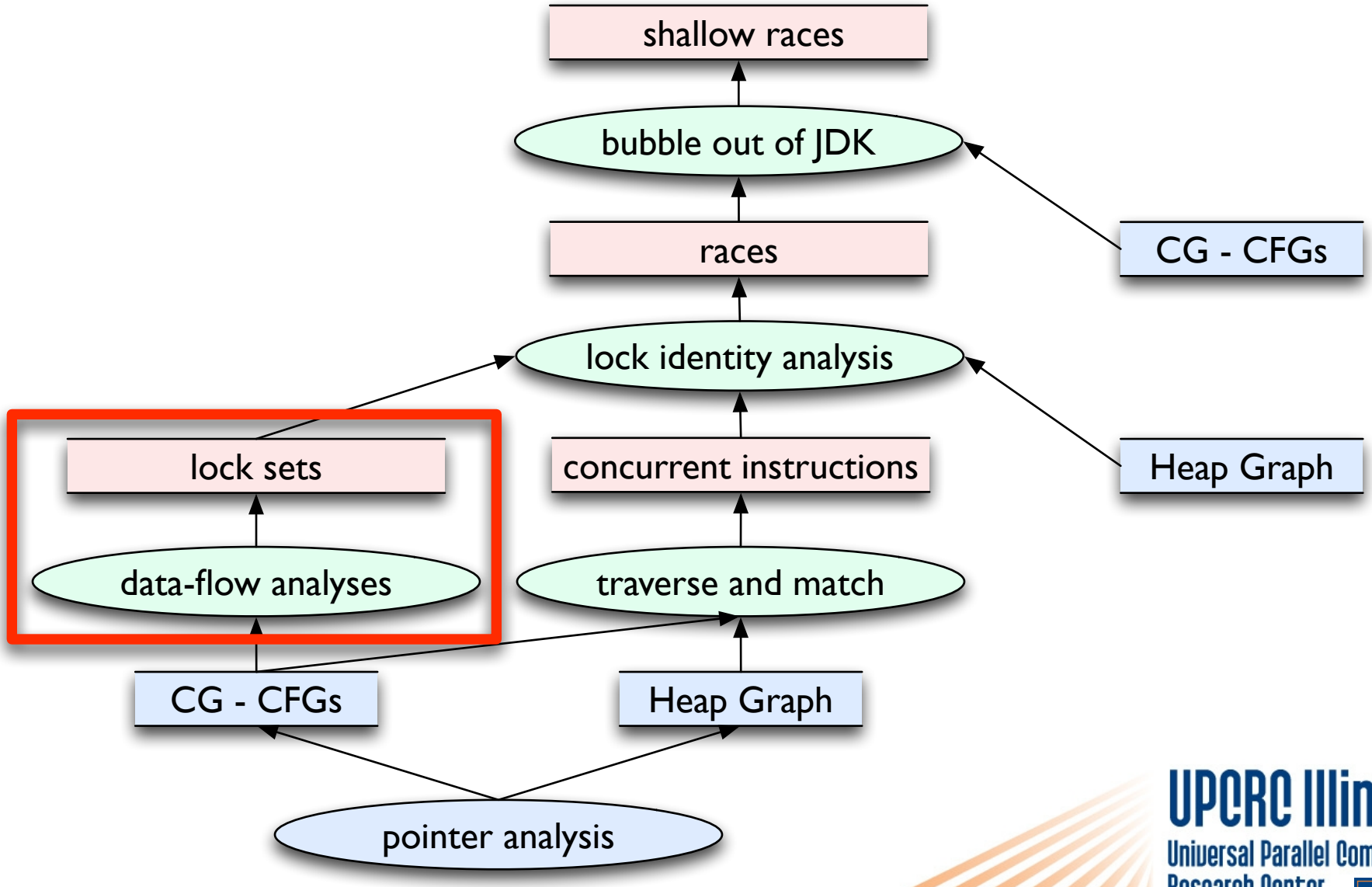
Concurrent Instructions

1. find all reads and writes in the parallel loops

2. match them based on aliases

=> pairs of instructions that can race

System overview



UPCRC Illinois
Universal Parallel Computing Research Center

Lock sets

```
synchronized(lock) {
    foo();
}
foo();
```

```
void foo() {
    shared.x++;
}
```

Inter-procedural data-flow analysis is needed

Lock sets analysis

Intra-procedural data-flow analysis

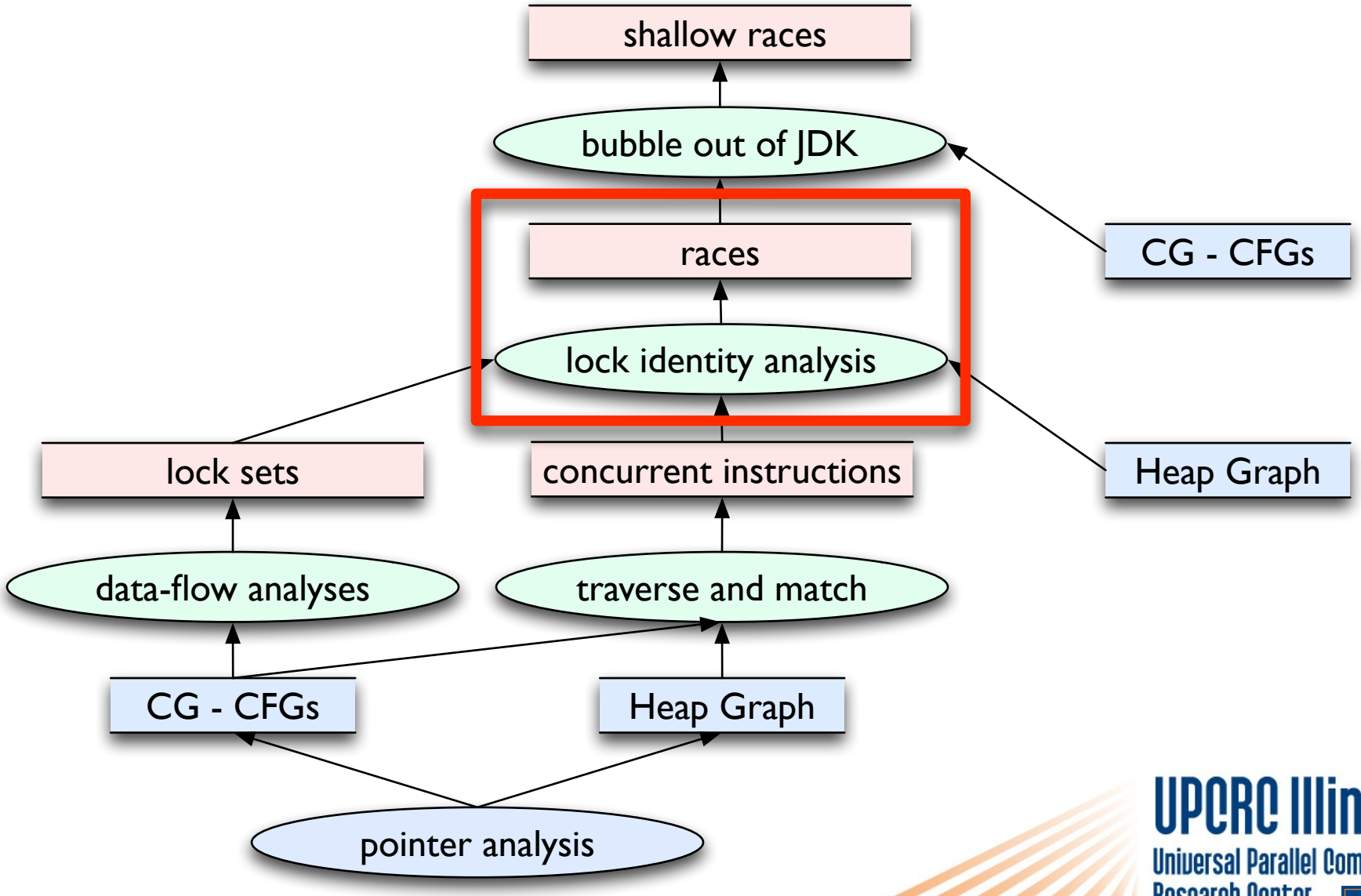
- over each method's CFG
- the lattice is the lock set power set

Inter-procedural data-flow analysis

- over call graph
- transfer functions on edges

  (call site to call graph node)

System overview



Lock identity. Why?

O

```
... new Computation() ...

class Computation {
    // center of mass
    Particle c = new Particle();
    Object lock;
    void compute() {
        [...]
```

T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        synchronized(lock) {
        this.c = new Particle(); }
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

T_2

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        synchronized(lock) {
        this.c = new Particle(); }
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

Lock identity. Why?

O

```
... new Computation() ...

class Computation {
    // center of mass
    Particle c = new Particle();
    Object lock;
    void compute() {
        [...]
```

T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        synchronized(lock) {
        this.c = new Particle(); }
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

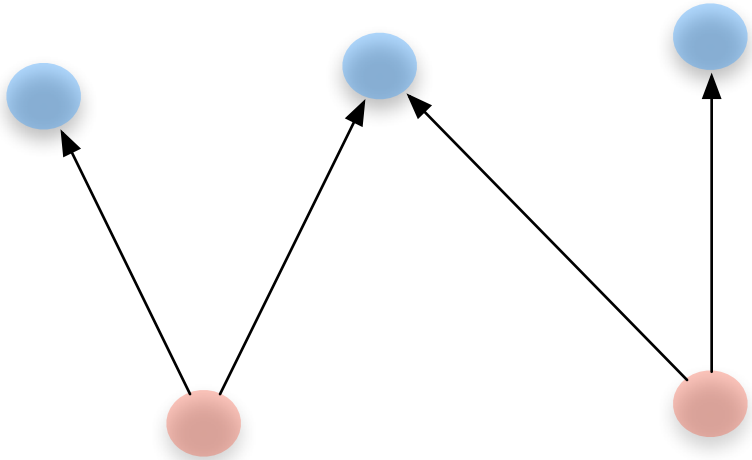
T_2

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        synchronized(lock) {
        this.c = new Particle(); }
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

Lock identity

may alias not enough

must alias needed



Lock identity

Conditional must-alias:

- represent each lock variable in a call graph node as a unique list of field dereferences from a reference that is know as unique, e.g. the array reference or a static field

Lock identity. Why?

O

```
... new Computation() ...

class Computation {
    // center of mass
    Particle c = new Particle();
    Object lock;
    void compute() {
        [...]
```

"this.lock"

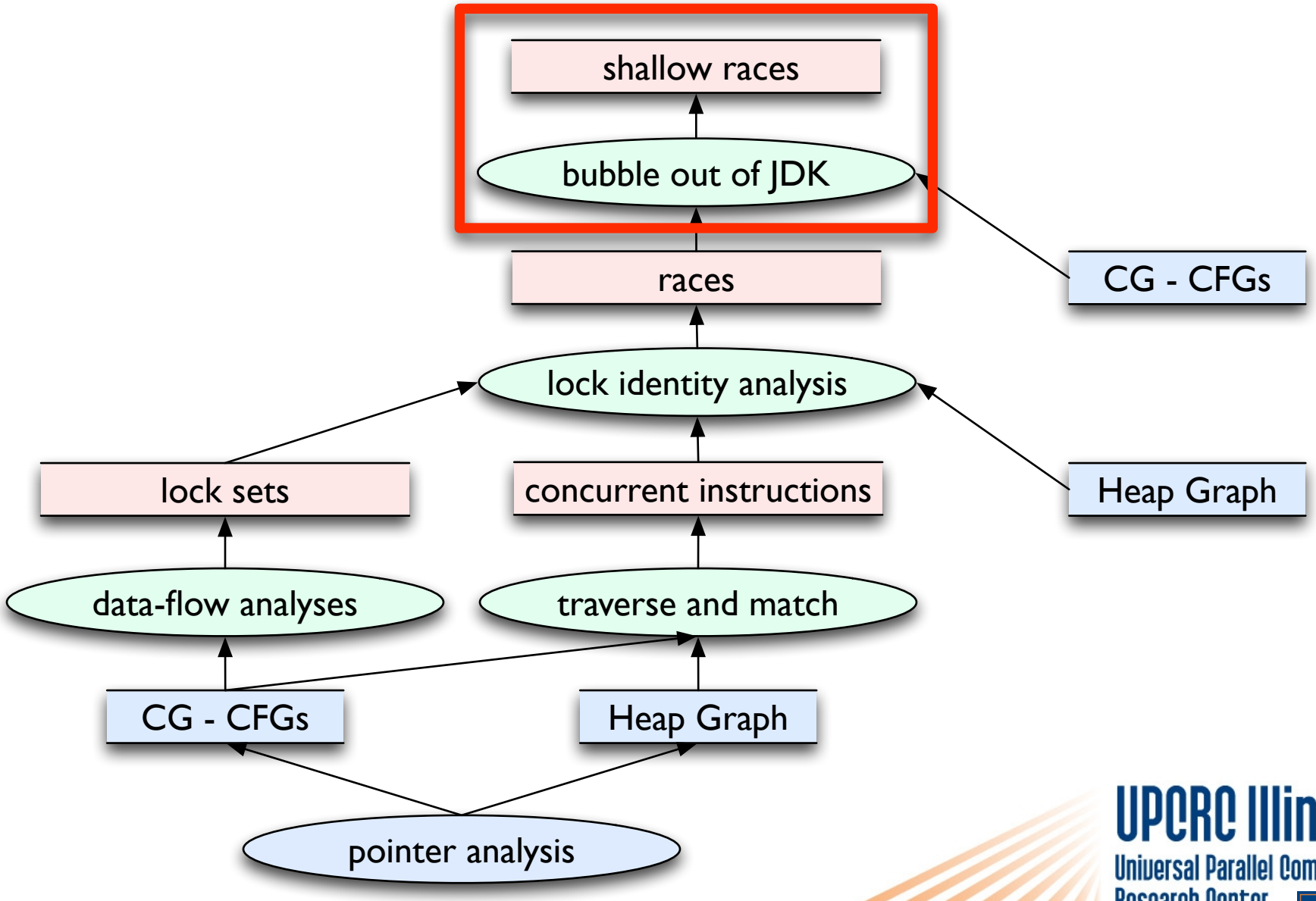
T_1

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        synchronized(lock) {
        this.c = new Particle(); }
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

T_2

```
particles.apply(...
    public void op(Particle p1) {
        Particle oldC = c;
        synchronized(lock) {
        this.c = new Particle(); }
        c.m = oldC.m + p.m;
        c.x = (oldC.x * ...
        c.y = (oldC.y * ...
    }
});
```

System overview



Motivating example

O

```
class Computation {
    HashSet results = new HashSet();

    void compute() {

        [...]
```

T_1

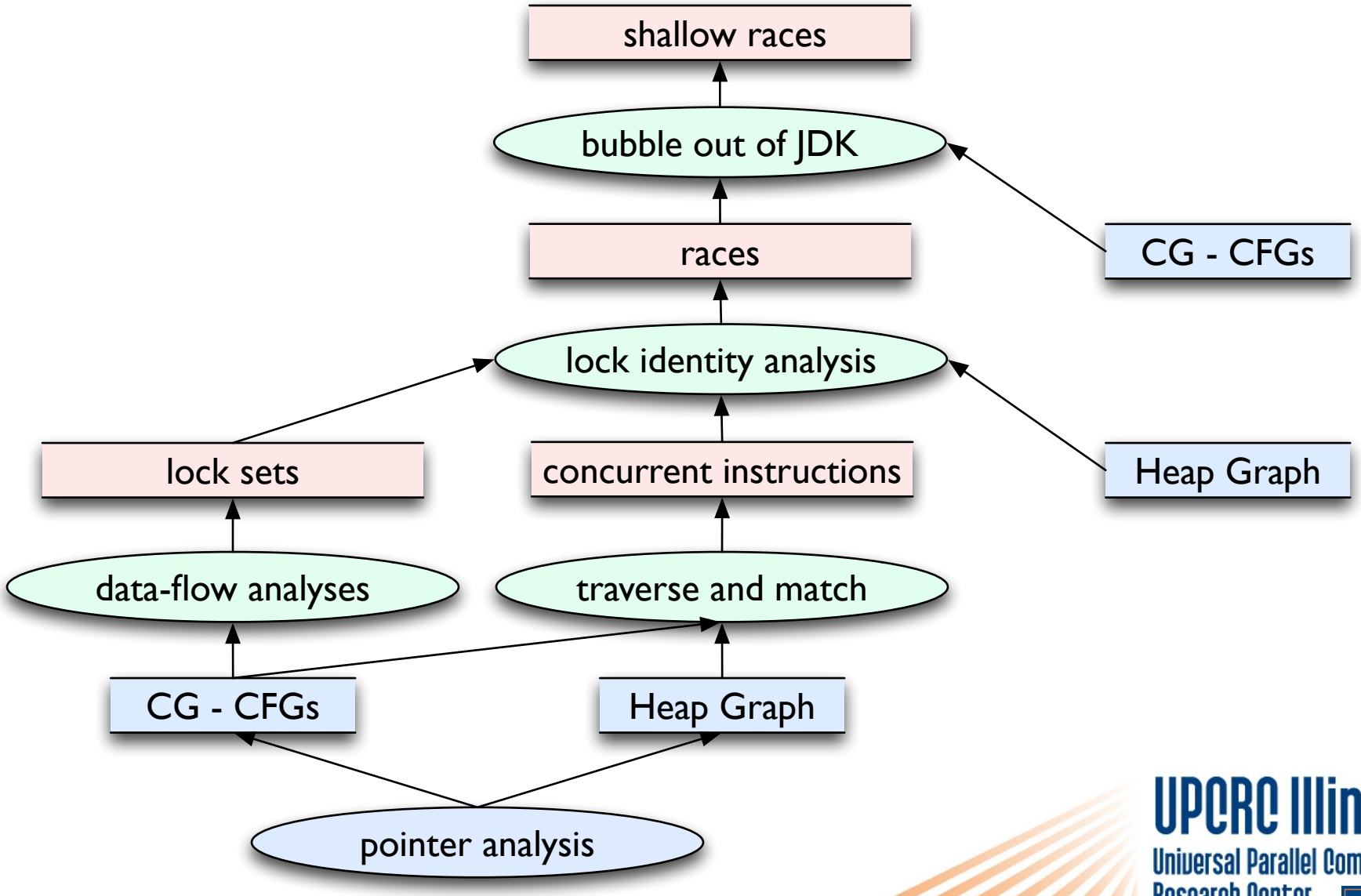
```
particles.apply(...
    public void op(Particle p1) {
        results.add(p1);
    }
});
```

T_2

```
particles.apply(...
    public void op(Particle p1) {
        results.add(p1);
    }
});
```

```
... in HashMap ...
Entry[] newTable = new Entry[newCapacity];
transfer(newTable);
table = newTable;
...
```

System overview



UPCRC Illinois
Universal Parallel Computing Research Center

Sources of unsoundness

classic, reflection and native code

other threads, aside from ParallelArray

... their existence can be easily checked

ParallelArray.getArray - shuffling the elements of the array outside the parallel array operators

... can easily be checked

Evaluation: Research questions

Is the analysis effective?

- total number of reported races
- false races
- missed races

Is the analysis efficient?

- runtime of the analysis

Evaluation methodology - subjects

select programs that use ParallelArray

... still few to none

Evaluation methodology - subjects

1. select programs with algorithms that have potential for loop parallelism

2. parallelize them using ParallelArray

Evaluation methodology - compare

state of the art static race detection for java - JChord

Effective Static Race Detection for Java. Mayur Naik, Alex Aiken, and John Whaley.

PLDI'06: ACM Conference on Programming Language Design and Implementation

Conditional Must Not Aliasing for Static Race Detection. Mayur Naik and Alex Aiken.

POPL'07: ACM Symposium on Principles of Programming Languages

Evaluation results

	Size (SLOC)			JChord				ReLooper				
		JDK		Analysis	Races		Missed	Analyzed	Analysis	Races		Missed
Project name	app	(filtered)	Bugs	time (s)	reported	real	bugs	invocations	time (s)	reported	real	bugs
Barnes-Hut sim.	899	220k	0	3m:05s	108	0	0	3197	20s	0	0	0
EM wave prop.	181	220k	0	1m:56s	16	0	0	2339	14s	0	0	0
MonteCarlo sim.	1441	220k	2	3m:10s	249	0	2	1473	12s	2	2	0
Coref	41k	220k	2	8m:59s	4622	0	2	11556	43s	80	80	0

UPCRC Illinois
Universal Parallel Computing Research Center

Conclusion

static race detection is useful

but a general one either:

- reports many false positives, or
- doesn't scale

our analysis is specialized for loop parallelism

=> better results