facebook

~~Reflex~~ Skip

Julien Verlaguet

Skip the things you've already computed!

Skip

A reactive programming language

Skip is a language with spreadsheet semantics. The runtime keeps track of external dependencies: when something changes, things “magically” update efficiently.

Skip

Why?

- Tools
- UI
- The skip compiler!
- Caches

Skip the Language

Language principles

- Favor immutability at function boundaries
- Reactivity is implicit
- Skip is embeddable!

Skip the Language

Language features

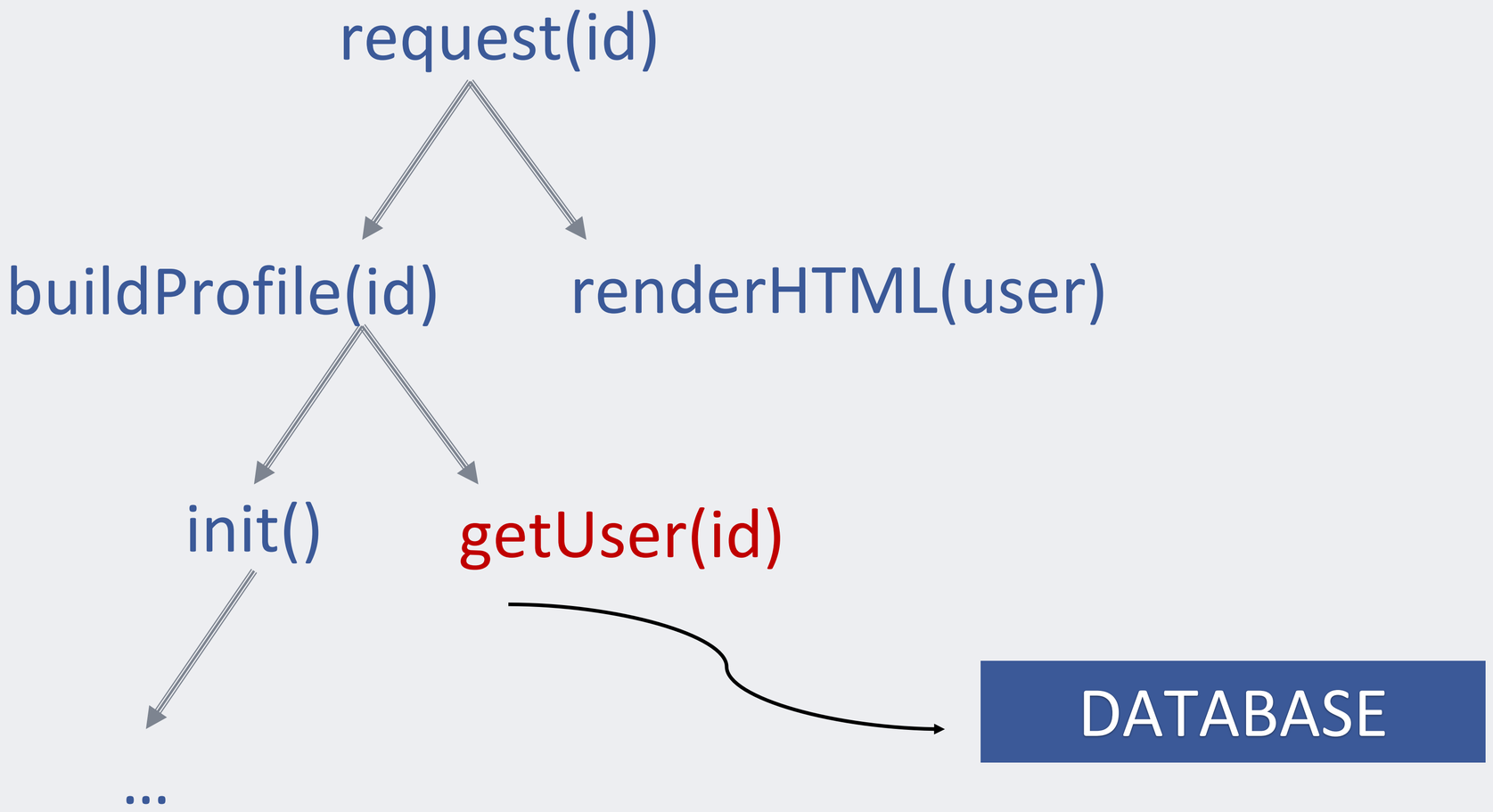
- Statically typed
- Generics
- Pattern-matching
- Closures
- Async/await
- Traits (type classes)
- …

Example

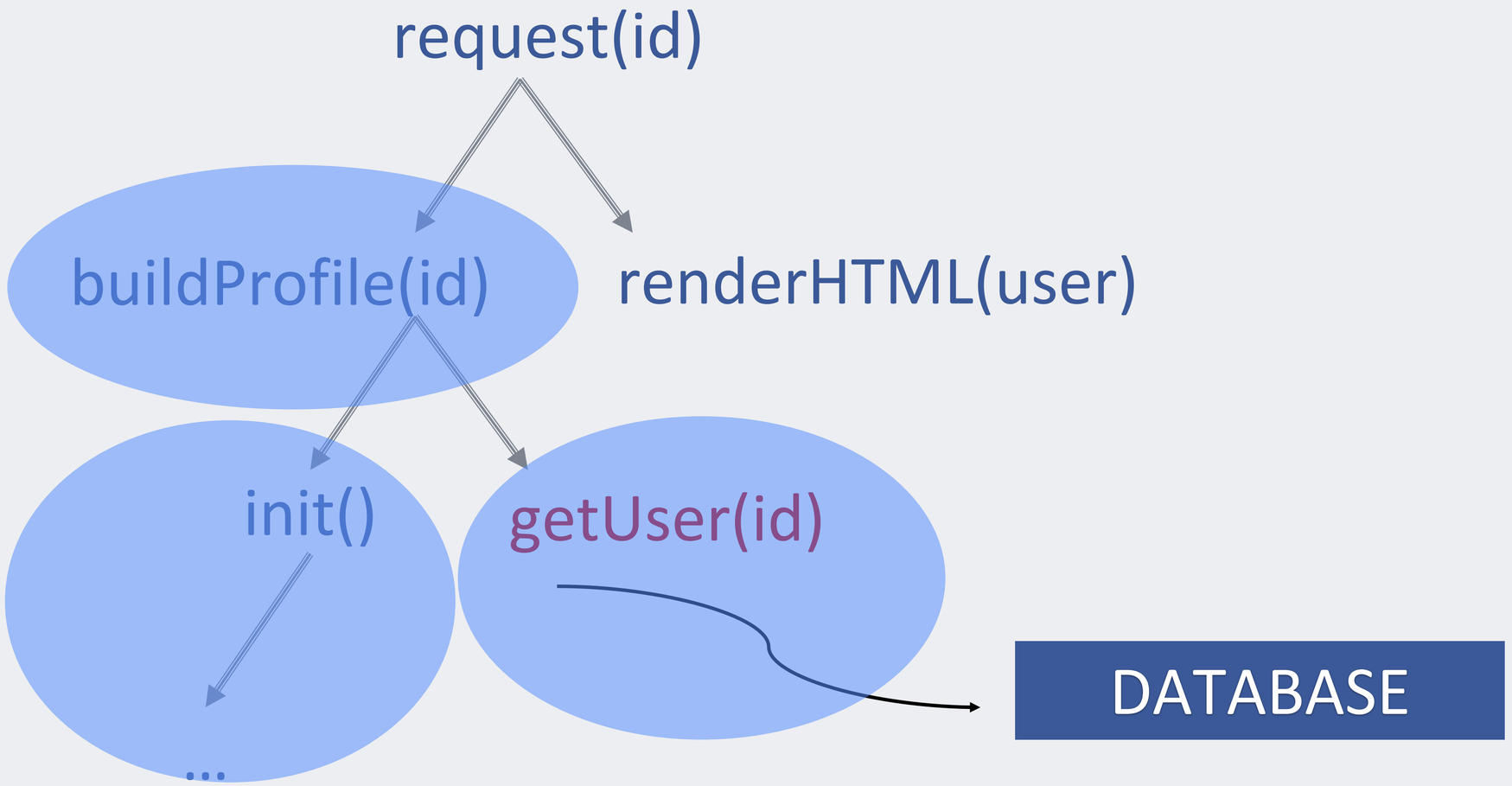
Skip: example

```
fun request(id: Int): String {
  user = buildProfile(id);
  renderHTML(user)
}
```

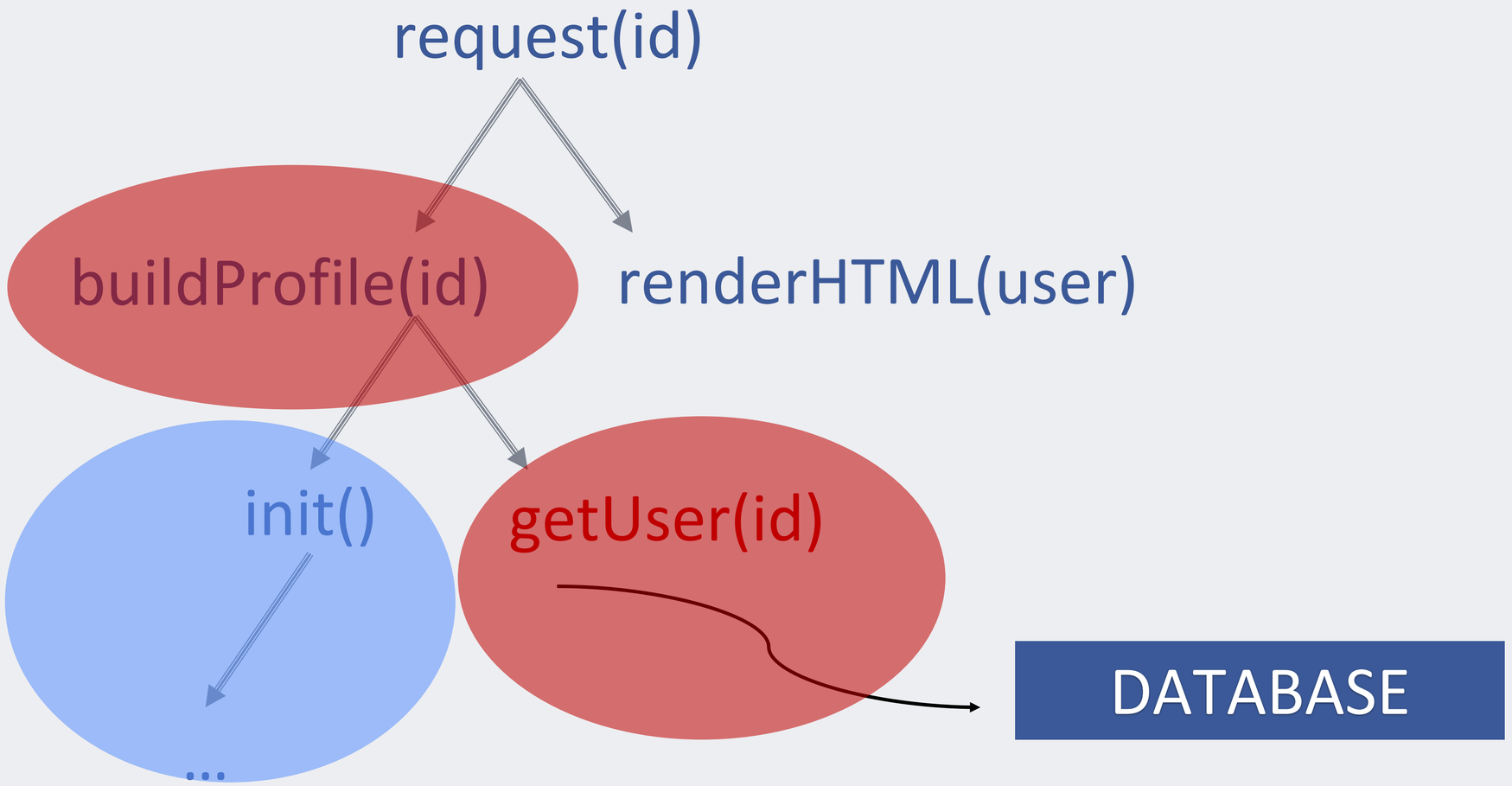
Skip: call graph



Skip: caching



Skip: invalidation



Skip: reactivity

```
fun request(id: Int): String {
  user = buildProfile(id);
  renderHTML(user)
}
```

Skip: mutation

```
fun request(id: Int): String {
  user = buildProfile(id);
  renderHTML(user)
}
```

MUTATION?

Skip's Solution

What invariants?

- **Immutability** guaranteed statically
  - *But local mutations are encouraged!*
- Runtime **tracks** dependencies
  - *Automatically invalidate memoized values*

Objects

Objects

```
class User(name: String, age: Int) {
  fun onBirthday(): User {
    User(name, this.age + 1)
  }
}
```

Objects

```
class User{name: String, age: Int} {
  fun onBirthday(): User {
    User{name, age => this.age + 1}
  }
}
```

Objects

```
class User{name: String, age: Int} {
  fun onBirthday(): User {
    !this.age = this.age + 1;
    this
  }
}
```

Objects

```
class User{name: …, mutable age: Int} {
  mutable fun onBirthday(): void {
    this.!age = this.age + 1
  }
}
```

Object creation: immutable

```
user = User {
  name => “julien”,
  age => 35
};
```

Cannot access mutable methods!

Object creation: mutable

```
user = mutable User {
  name => “julien”,
  age => 35
};
```

Mutable methods until freezing!

Skip: collections

```
v1 = Vector[1, 2, 3];
v2 = mutable Vector[1, 2, 3];
m1 = Map[1, 2, 3];
m2 = mutable Map[1, 2, 3];
v1.filter(x -> x >= 0).map(x -> x + 1)
// Make your own!
```

Skip: collections

```
fun filterUsers(): SMap<String> {
  result = mutable Map[];
  …
  freeze(result)
}
```

OO = FP

Skip: pattern-matching

```
fun isNull(value: ?X): Bool {
  value match {
   | Null -> true
   | Box(x) -> x
  }
}
```

OO Inheritance

base class IntList

class Nil() extends IntList

class Cons(Int, IntList) extends IntList

FP inheritance

```
base class IntList {
  children =
  | Nil()
  | Cons(Int, IntList)
}
```

OO methods

```
class Nil() extends IntList {
  fun add1(): IntList { … }
}

class Cons(Int, IntList) {
  fun add1(): IntList { … }
}
```

FP methods

```
base class IntList {
  ...
  fun add1(): IntList
  | Nil() -> ...
  | Cons(x, rl) -> ...
}
```

OO/FP mix

```
base class IntList {
  fun add1(): IntList
  | Nil() -> …
}

class Cons(Int, IntList) extends IntList {
  fun add1(): IntList { … }
}
```

Generics

Generics

```
base class List<T> {
  children =
  | Nil()
  | Cons(T, List<T>)

  fun map<T2>(f: T -> T2): List<T2>
  | Nil() -> ...
  | Cons(x, rl) -> ...
}
```

Generics

```
base class List<T> {


  fun show[T: Showable](): String {
    // ..
  }
}
```

Generics: traits

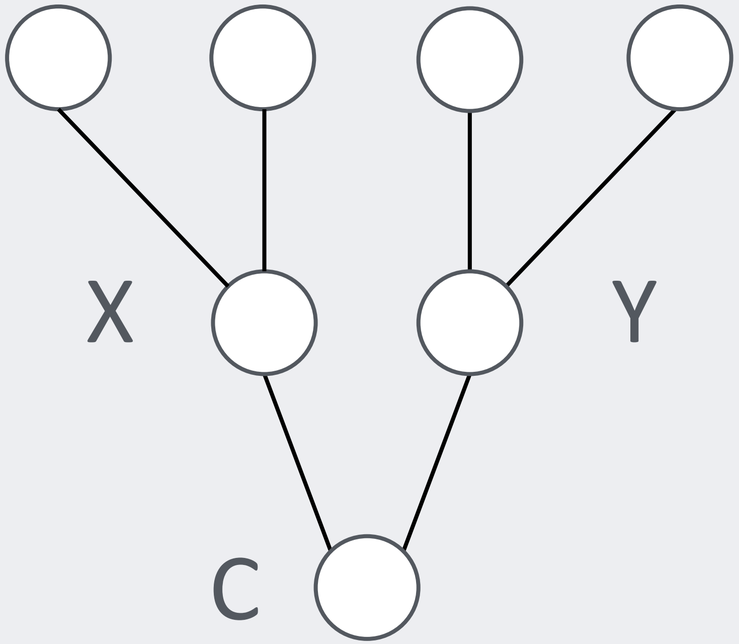
```
trait Showable {


  fun toString(): String {
    // ..
  }
}
```

Lvalues

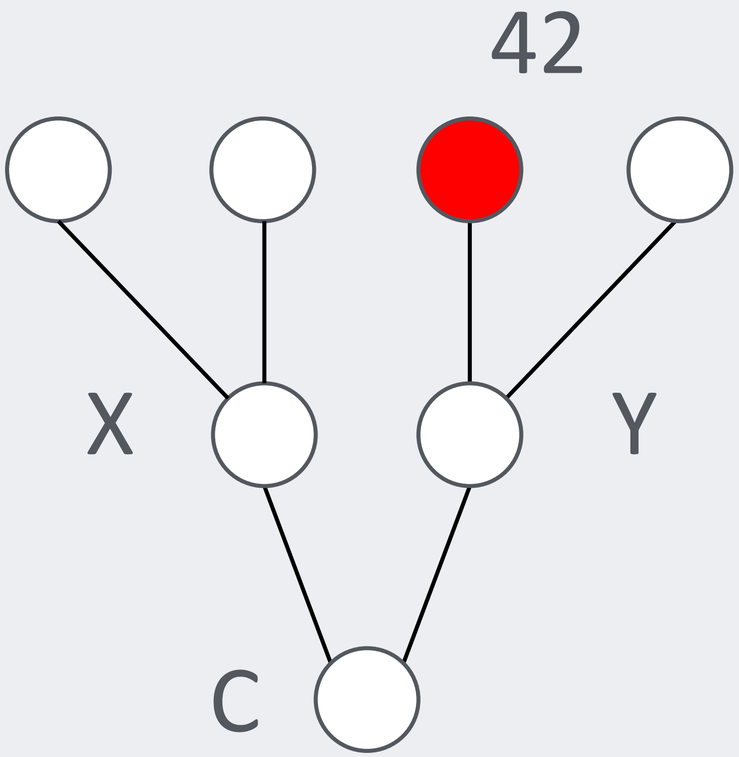
Lvalues

local = C(X(1, 2), Y(3, 4));



Lvalues

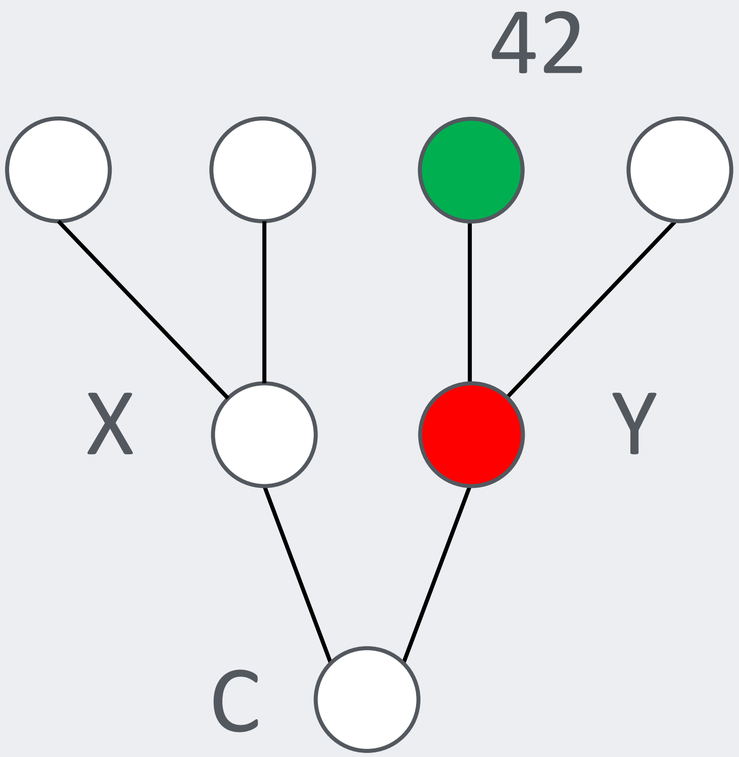
local.field2.!field1 = 42;



Modified

Lvalues

local.!field2.field1 = 42;

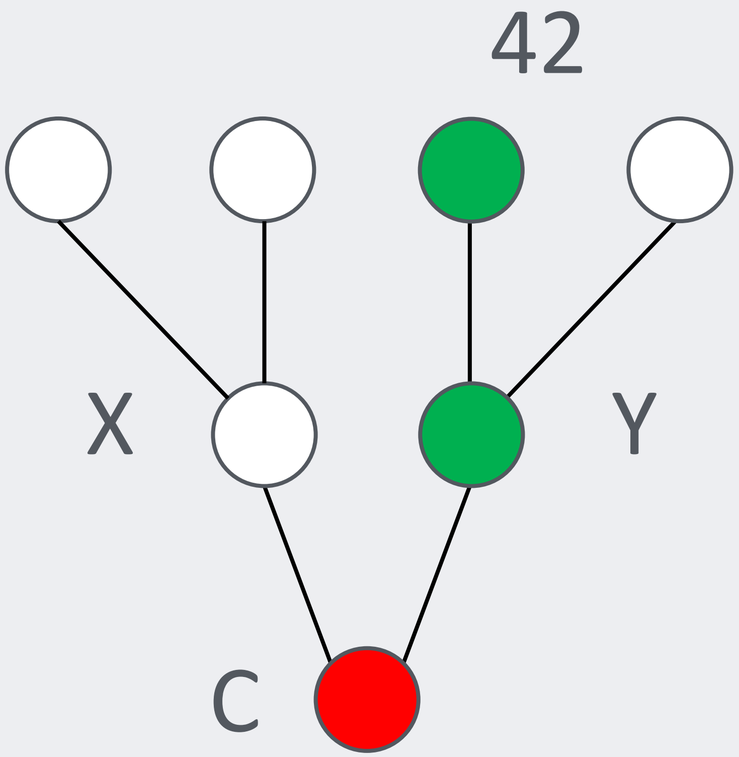


New value

Modified

Lvalues

!local.field2.field1 = 42;



New value

Modified

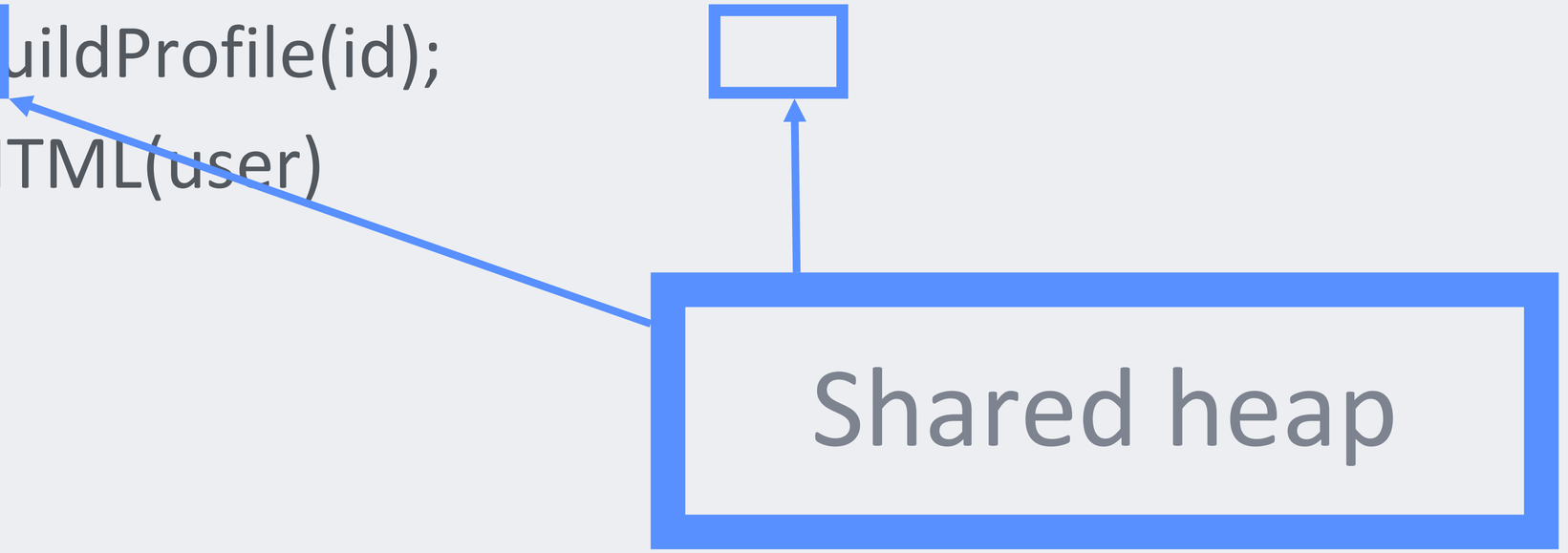
Code: lvalues

(!acc, result) = some_fun(acc, x);

Memoization

Skip: memoization is fast!

```
fun request(id: Int): String {
  user = buildProfile(id);
  renderHTML(user)
}
```



Shared heap

- Objects are immutable
- 2 structurally equal object have the same address
- Supports multithreading (MVCC)

Skip: memoization is fast!

```
fun request(id: Int): String {
  user = buildProfile(id);
  user = buildProfile(id);
  renderHTML(user)
}
```

INSTANT

Skip: memoization is super fast!

```
fun request(id: Int): String {
  user = buildProfile(id);
  _ = renderHTML(user);
  renderHTML(user);
}
```

INSTANT

Memory model

Memory model

- Skip is safe, garbage-collected
- Minor heap
  - Fast, “bump a pointer” allocator
- Shared heap
  - Long-lived, immutable
  - Unique (“interned”) objects only

Skip: the @gc annotation

```
@gc fun request(id: Int): String {
  user = buildProfile(id);
  renderHTML(user)
}
```

Memory model properties

- O(allocated) != O(heap_size)
- Annotations to memoize
- You can reason about performance!

Future applications

Future applications

Time shift

- Compute in the morning, use in the afternoon
- Eases “At peak” traffic

Future applications

Remote cache

- Use local data as if it was remote
- Maintaining a reactive version
- Push changes

Thank you!

Example

Slide Subtitle

Runtime

Slide Subtitle

- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam
- Pellentesque in nibh quis est cursus fermentum.
- Donec vel justo ut erat hendrerit nec non leo.
- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam

Applications

Slide Subtitle

- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam
- Pellentesque in nibh quis est cursus fermentum.
- Donec vel justo ut erat hendrerit nec non leo.
- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam

Slide Title

Slide Subtitle

- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam
- Pellentesque in nibh quis est cursus fermentum.
- Donec vel justo ut erat hendrerit nec non leo.
- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam

Slide Title

Slide Subtitle

- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam
- Pellentesque in nibh quis est cursus fermentum.
- Donec vel justo ut erat hendrerit nec non leo.
- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam

Slide Title

Slide Subtitle

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed semper mauris quis sem finibus, a feugiat eros sodales. Aliquam vel turpis eleifend, suscipit tortor venenatis, luctus metus. Phasellus lorem massa, venenatis quis maximus ut, consectetur vel libero. Phasellus lorem massa.

Slide Title

Slide Subtitle

- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam
- Pellentesque in nibh quis est cursus fermentum.
- Donec vel justo ut erat hendrerit nec non leo.
- Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam

Slide Title

- Lorem ipsum dolor sit amet, consectetur.
  - Donec vitae ipsum quis elit ultricies aliquam
    - Pellentesque in nibh quis est cursus fermentum.
      - Donec vel justo ut erat hendrerit nec non leo.
        - Lorem ipsum dolor sit amet, consectetur.
- Donec vitae ipsum quis elit ultricies aliquam

Interstitial Title

“Lorem ipsum dolor sit amet,
adipiscing elit. Nam facilisis erat nunc, maximus auctor
libero ornare at. Ut nec nisl sodales quam feugiat.”

— Quote Author

Source: sed ut unde omnis

Slide Title

Slide Subtitle

<code></code>
<code></code>
<code></code>
<code></code>
<code></code>
<code></code>
<code></code>
<code></code>
<code></code>

facebook