جامعة نيويورك أبوظبي

NYU ABU DHABI

IDE Framework

CS-UH 3260
Static Program Analysis

Karim Ali
@karimhamdanali

Previously

- IFDS
- Graph reachability
- Path/summary edges
- Large domains => low performance

Other Issues with IFDS

- Complexity of IFDS is O(|E|・|D|3)
- Encoding is not efficient for some analyses
- Merge is restricted to set union

Example: Linear Constant Propagation

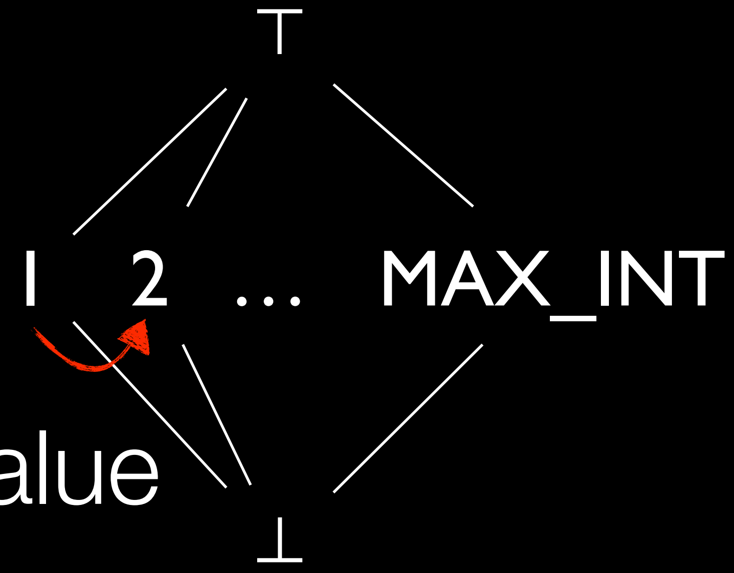
- Can be encoded in IFDS
- As abstraction, use pairs (var, value)
  - where value is an element of a (finitely bounded) set of integers

Problem 1: Domain D is quite large…

- Remember: merge has to remain set union

Problem 2: Union does not discard any values

Issues with LCP & IFDS



- Summaries grow with any new input value
  1. Domain is very broad
  2. Flow functions can move a value “sideways”
- Merge operator is union => cannot apply more sophisticated merges that would keep the domain small (e.g.,1⊔5 = [1..10])
- If MAX_INT= ∞, then LCP cannot be encoded as an IFDS problem!

Interprocedural (finite) Distributive Environment Problems

Thomas Reps, Susan Horwitz, and Mooly Sagiv. Precise Interprocedural Dataflow Analysis via Graph Reachability. ACM Transactions on Programming Languages and Systems (TOPLAS), 17(6), 1021–1071.

IDE

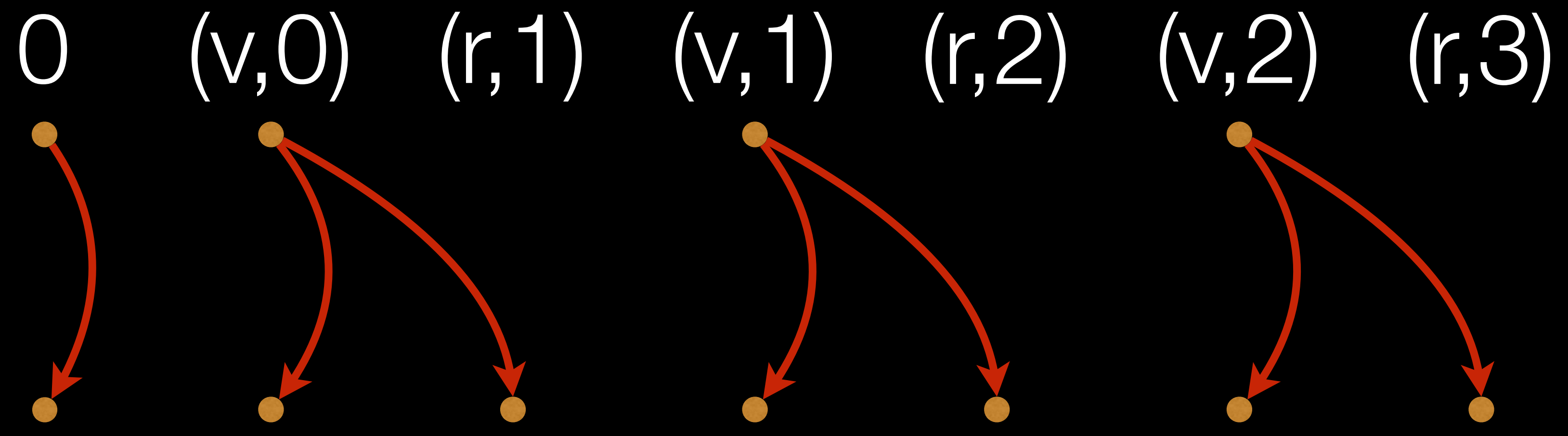
```
main() {
 x = inc(1);
 y = inc(x);
```

```
inc(v) {
 r = v + 1;
```

Summaries

0↦0 (v,i)↦(v,i) (v,i)↦(r,i+1)

```
}
```



```
inc(v) {
 r = v + 1;
 return r;
}
```

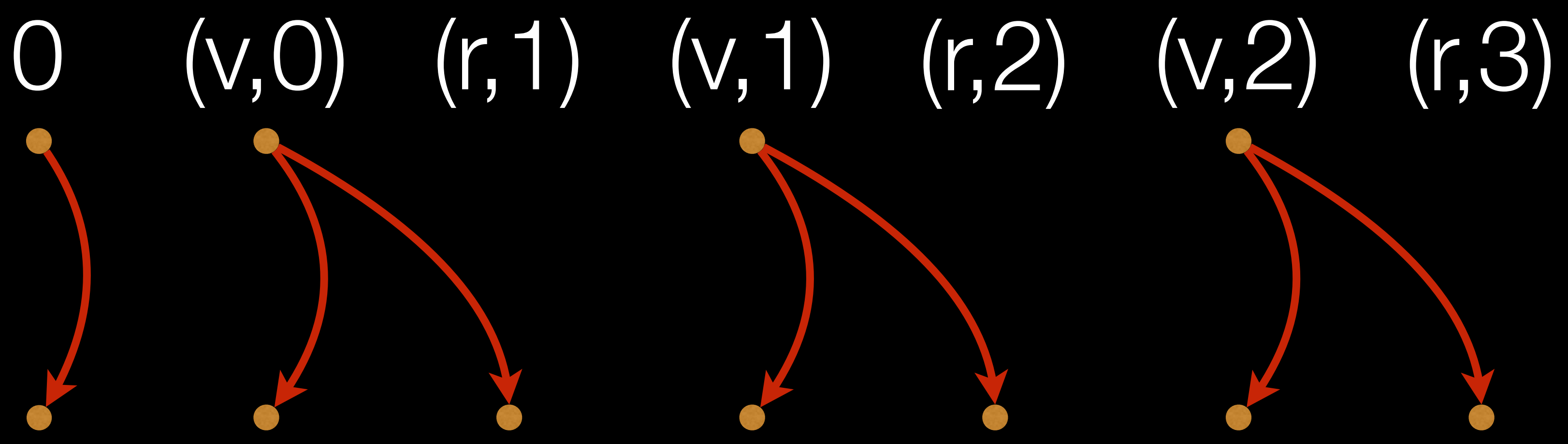
$(\underline{v},\underline{i})\mapsto(\underline{r},\underline{i+1})$

context-independent

- the same in every context
- know how to represent this: as edge v→r in IFDS

context-dependent

- while the value *i* is context-dependent, the function $i\mapsto i+1$ is not
- idea: annotate edge with function

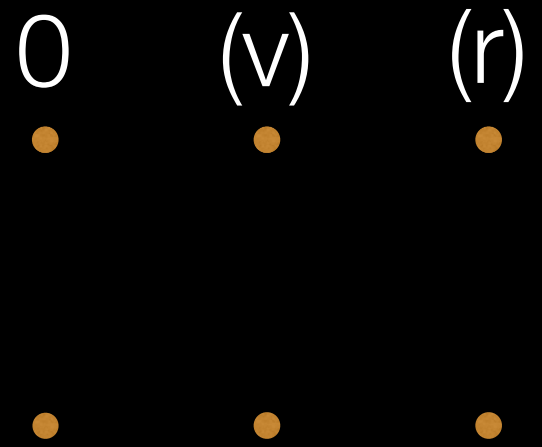


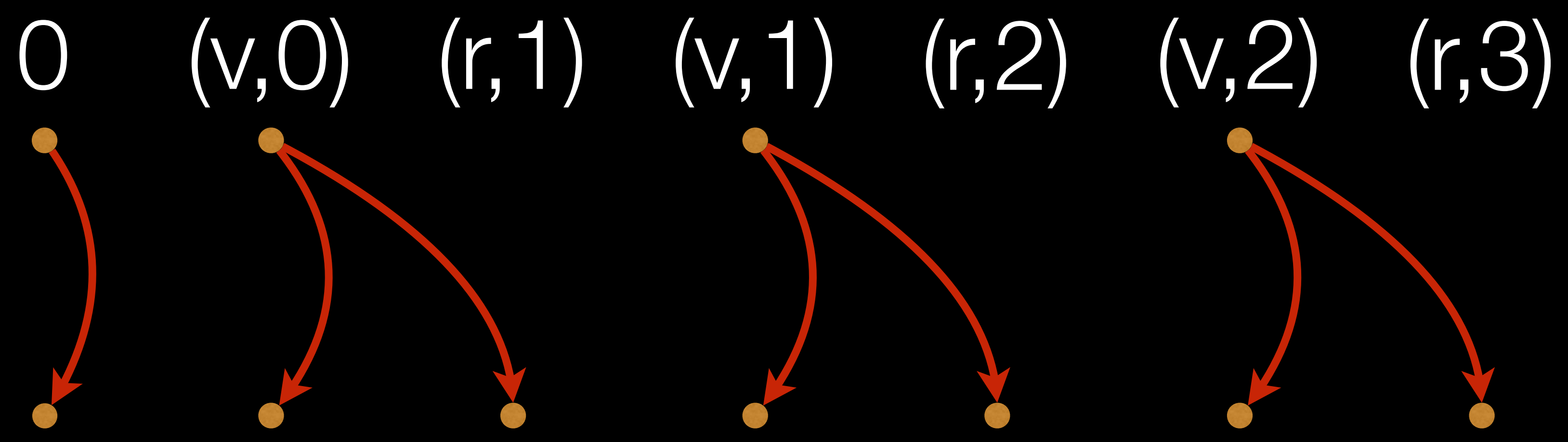
```
inc(v) {
 r = v + 1;
 return r;
}
```

Summaries	(v,0)↦(v,0)	(v,0)↦(r,1)
	(v,1)↦(v,1)	(v,1)↦(r,2)
0↦0	(v,2)↦(v,2)	(v,2)↦(r,3)

Summaries

$\underbrace{0}\mapsto\underbrace{0}$ $\underbrace{(v,i)}\mapsto\underbrace{(v,i)}$ $\underbrace{(v,i)}\mapsto\underbrace{(r,}i+1)$



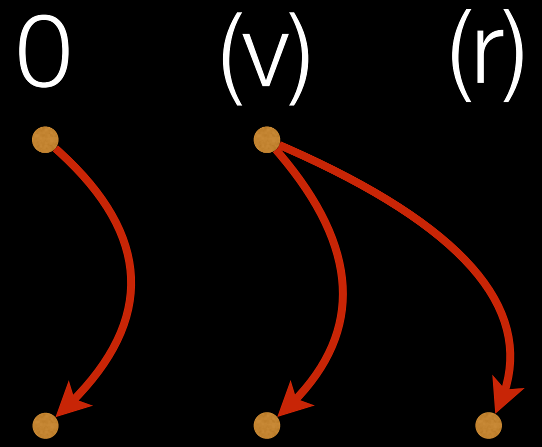


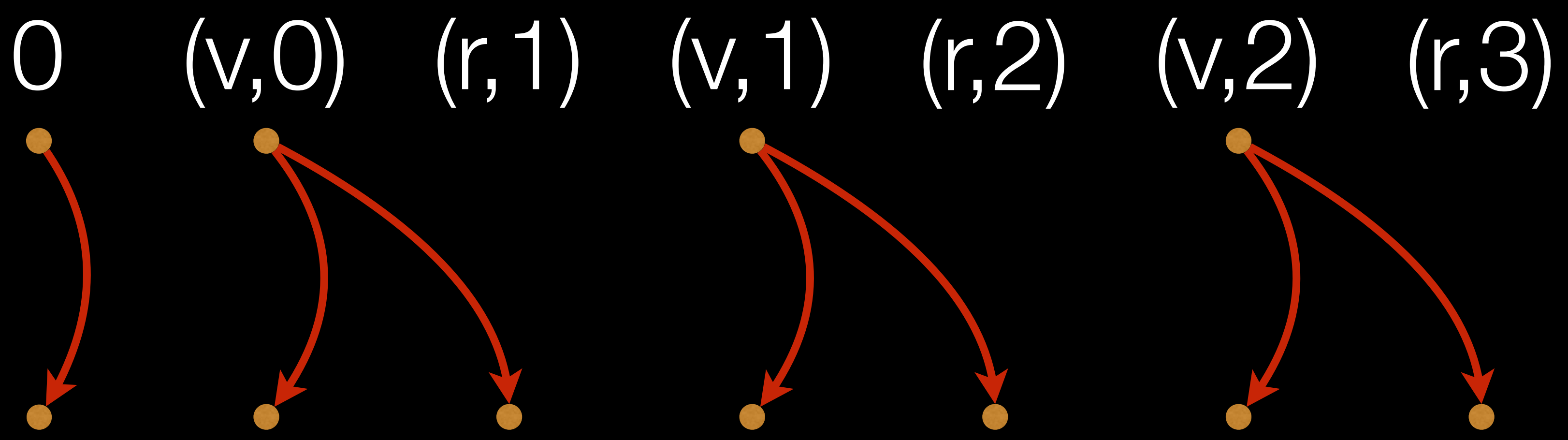
```
inc(v) {
 r = v + 1;
 return r;
}
```

Summaries	(v,0)↦(v,0)	(v,0)↦(r,1)
	(v,1)↦(v,1)	(v,1)↦(r,2)
0↦0	(v,2)↦(v,2)	(v,2)↦(r,3)

Summaries

$\underbrace{0}\mapsto\underbrace{0}$ $\underbrace{(v},i)\mapsto\underbrace{(v},i)$ $\underbrace{(v},i)\mapsto\underbrace{(r},i+1)$





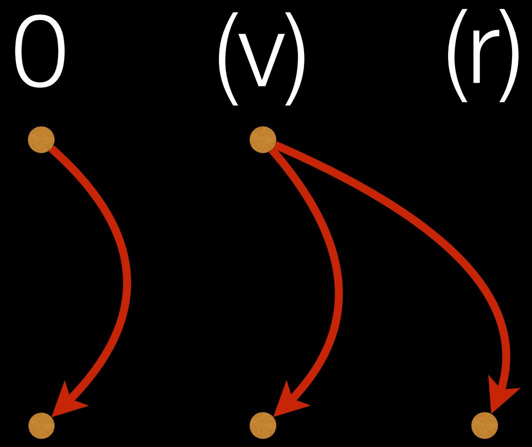
0 (v,0) (r,1) (v,1) (r,2) (v,2) (r,3)

```
inc(v) {
 r = v + 1;
 return r;
}
```

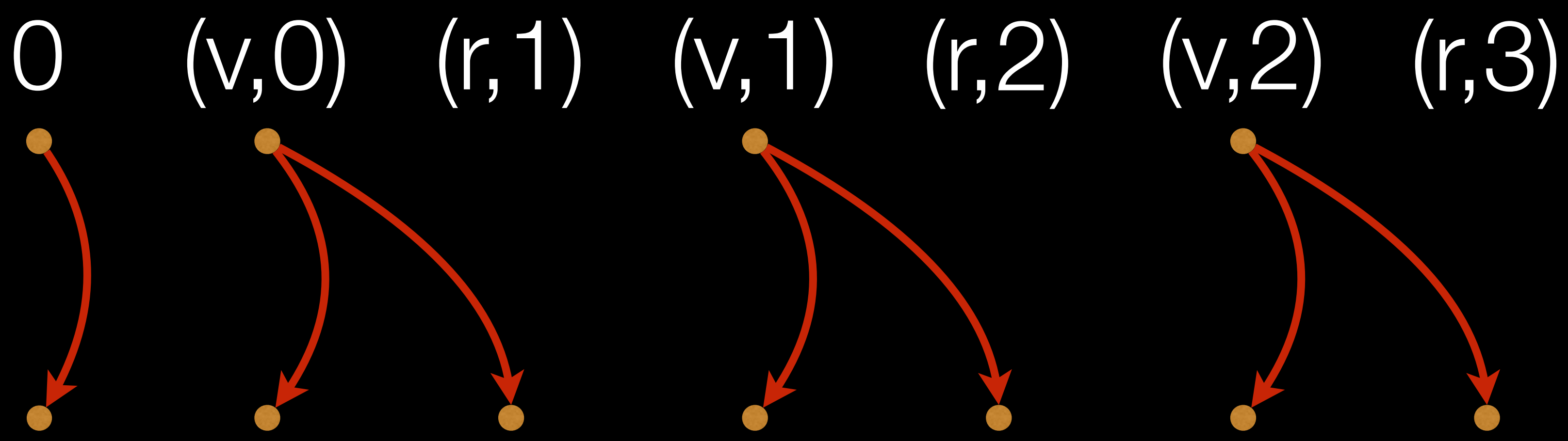
Summaries	(v,0)↦(v,0)	(v,0)↦(r,1)
	(v,1)↦(v,1)	(v,1)↦(r,2)
0↦0	(v,2)↦(v,2)	(v,2)↦(r,3)

Summaries

0↦0 $(v,\underline{i})\mapsto(v,\underline{i})$ $(v,\underline{i})\mapsto(r,\underline{i+1})$



0 (v) (r)



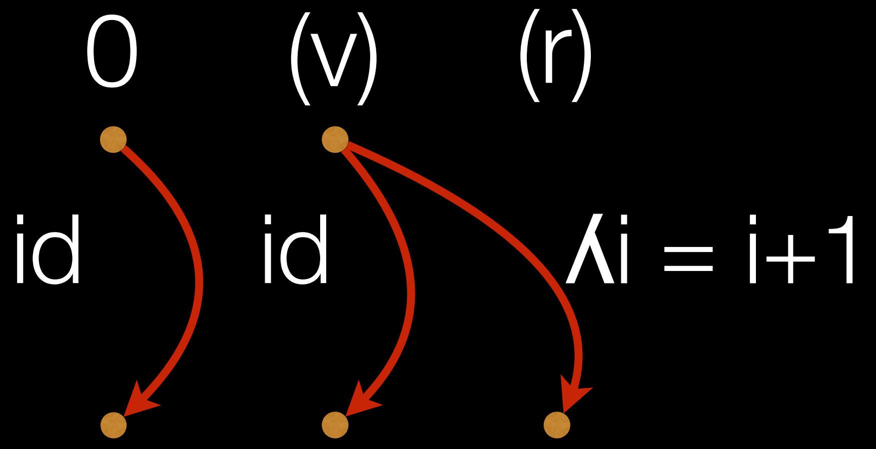
0 (v,0) (r,1) (v,1) (r,2) (v,2) (r,3)

```
inc(v) {
 r = v + 1;
 return r;
}
```

Summaries	(v,0)↦(v,0)	(v,0)↦(r,1)
	(v,1)↦(v,1)	(v,1)↦(r,2)
0↦0	(v,2)↦(v,2)	(v,2)↦(r,3)

Summaries

0↦0 $(v,\underline{i})\mapsto(v,\underline{i})$ $(v,\underline{i})\mapsto(r,\underline{i+1})$



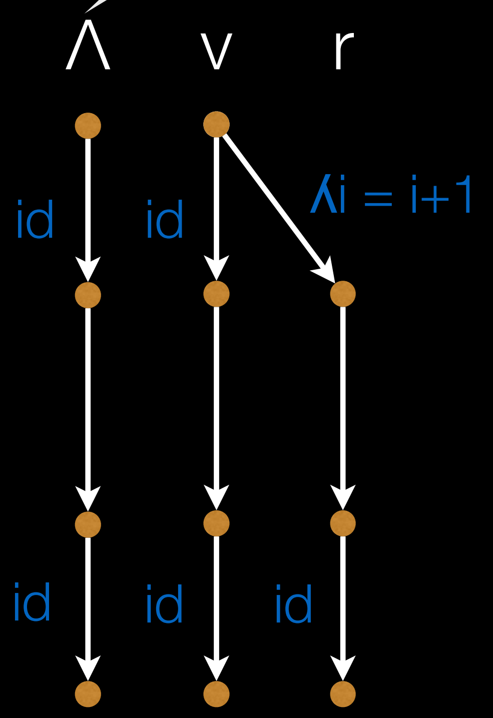
0 (v) (r)

id id λi = i+1

IDE Phases

Phase 1: context-independent information

Phase 2: context-dependent information



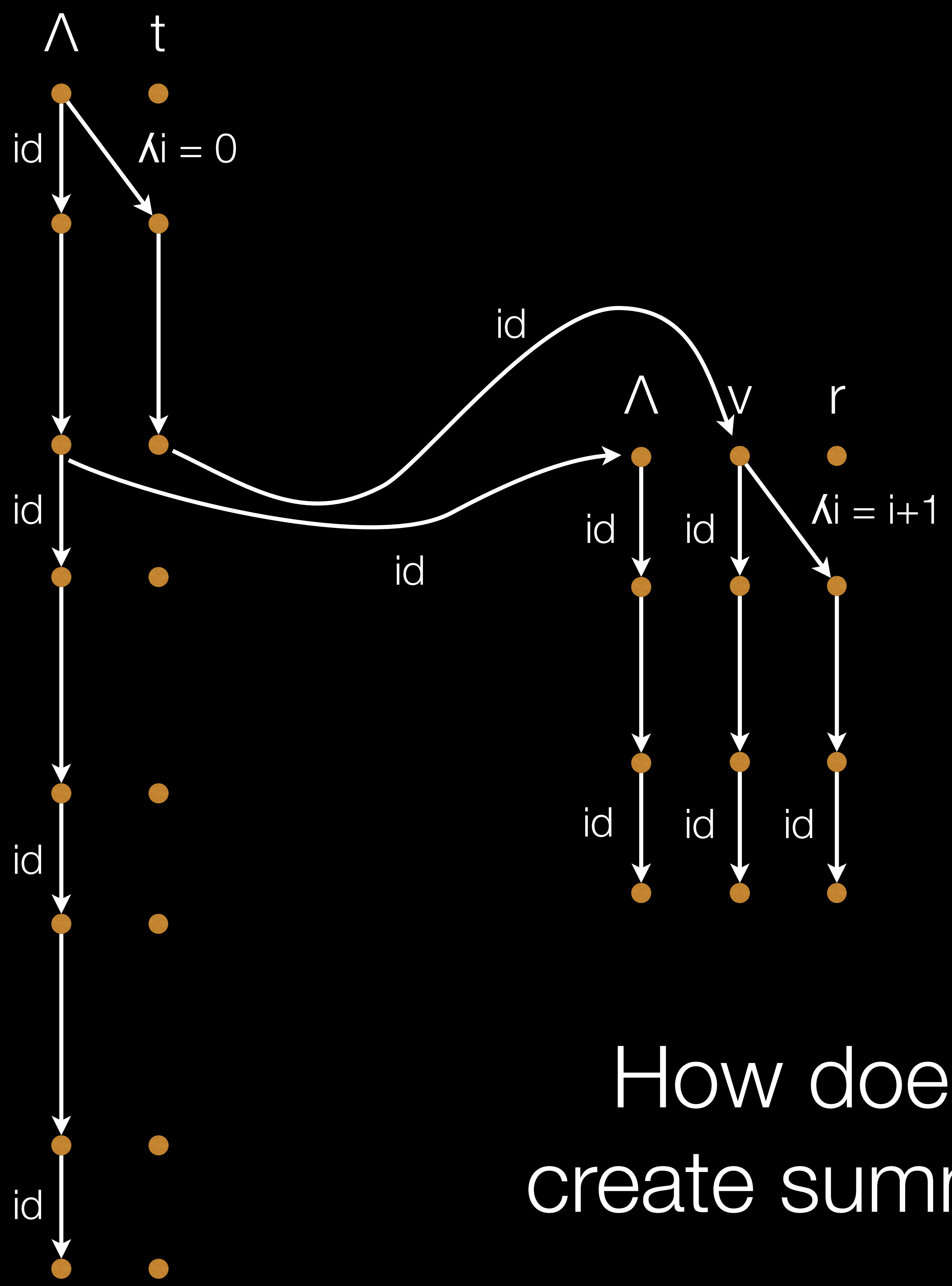
```
r = v + 1;
```

```
return r;
```

formerly known as 0 in IFDS

now encoded in a context-independent format

```
t = 0;

x = inc(t);

y = inc(x);

print(y);
```



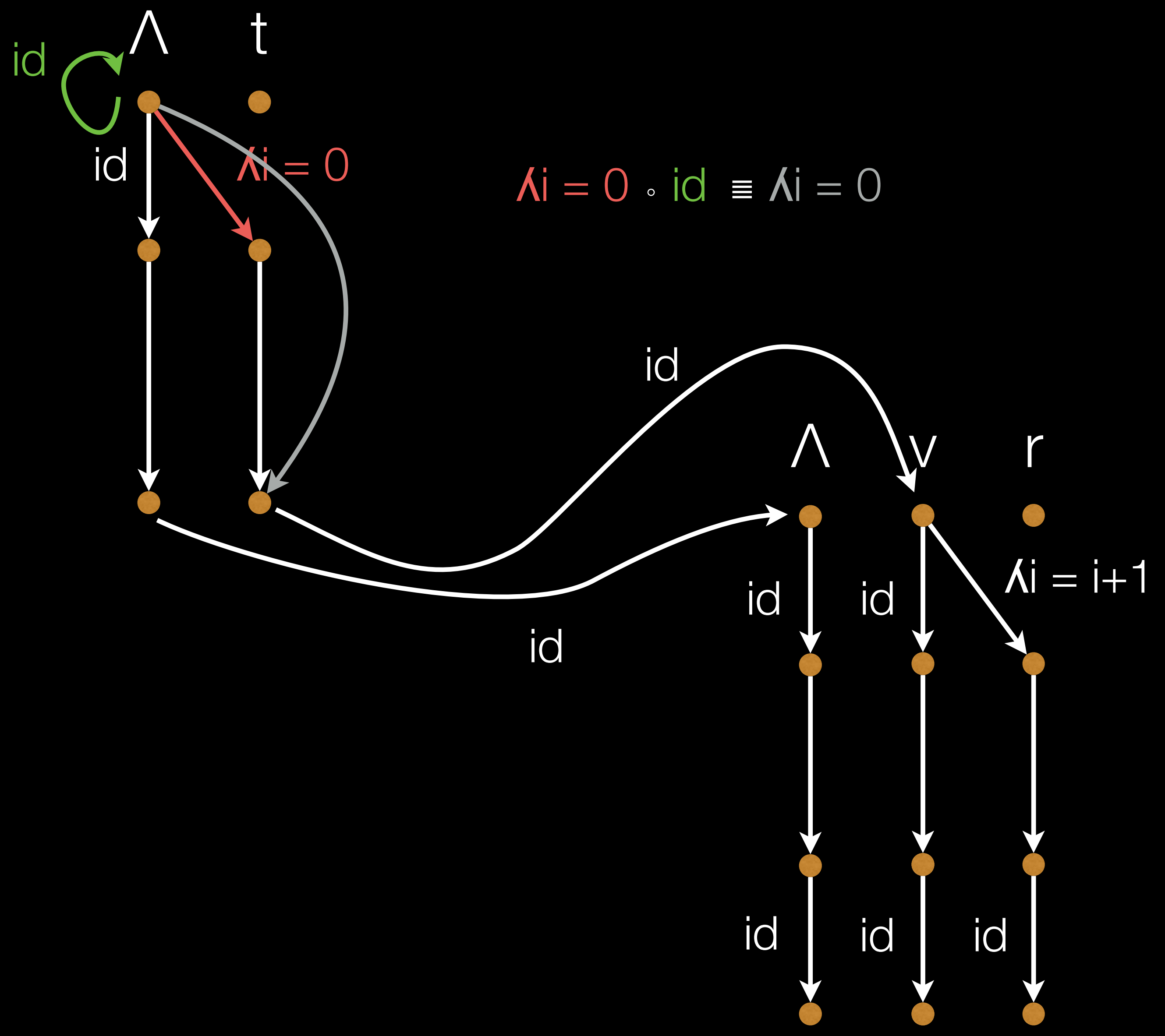
```
r = v+1;

return r;
```

How does IDE create summaries?

Jump Functions

```
t = 0;

x = inc(t);

y = inc(x);

print(y);
```

```
r = v+1;

return r;
```



$\lambda i = 0 \circ \mathrm{id} \equiv \lambda i = 0$

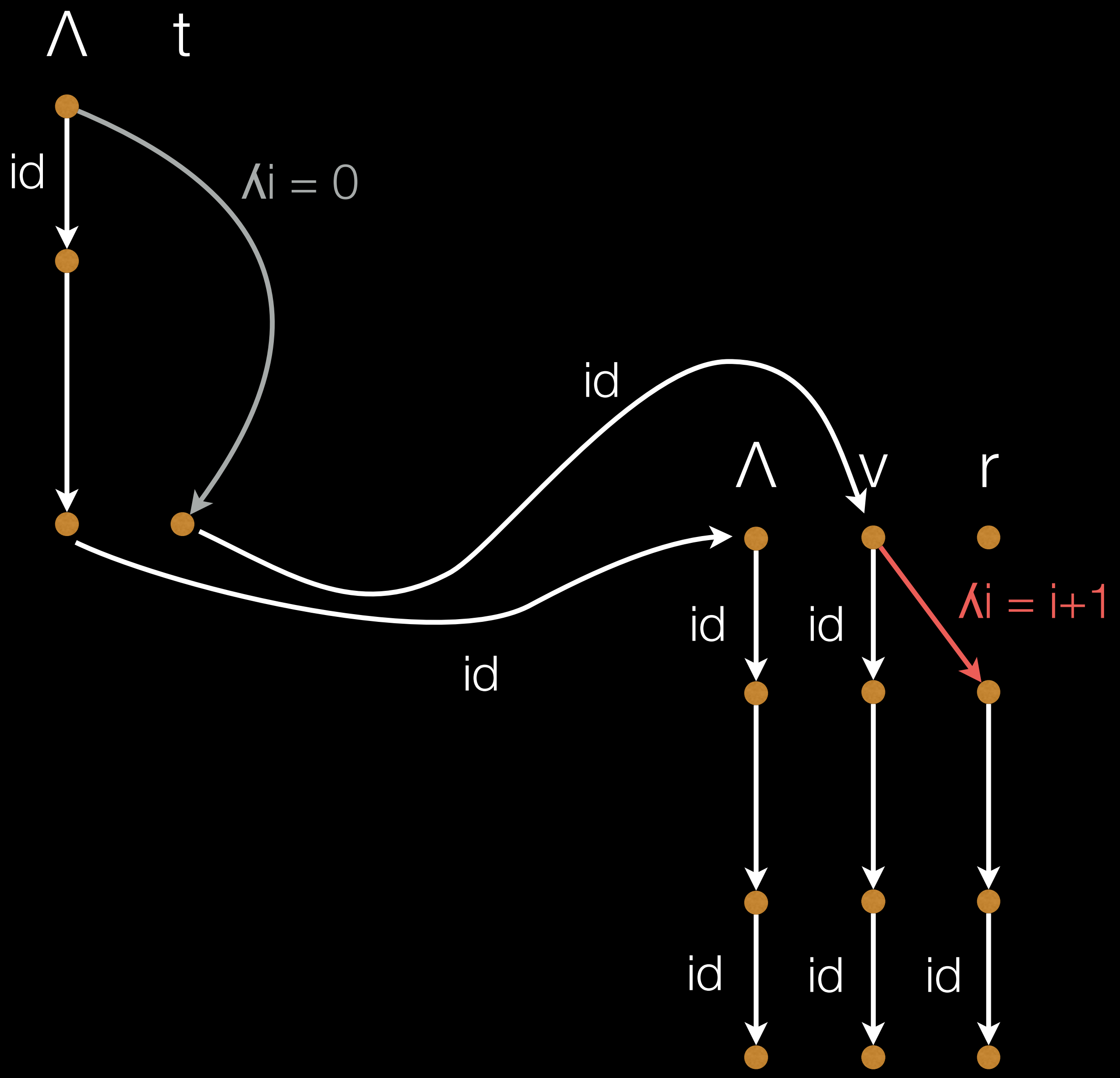
Jump Functions

```
t = 0;
```

```
x = inc(t);
```

```
y = inc(x);
```

```
print(y);
```



```
r = v+1;
```

```
return r;
```

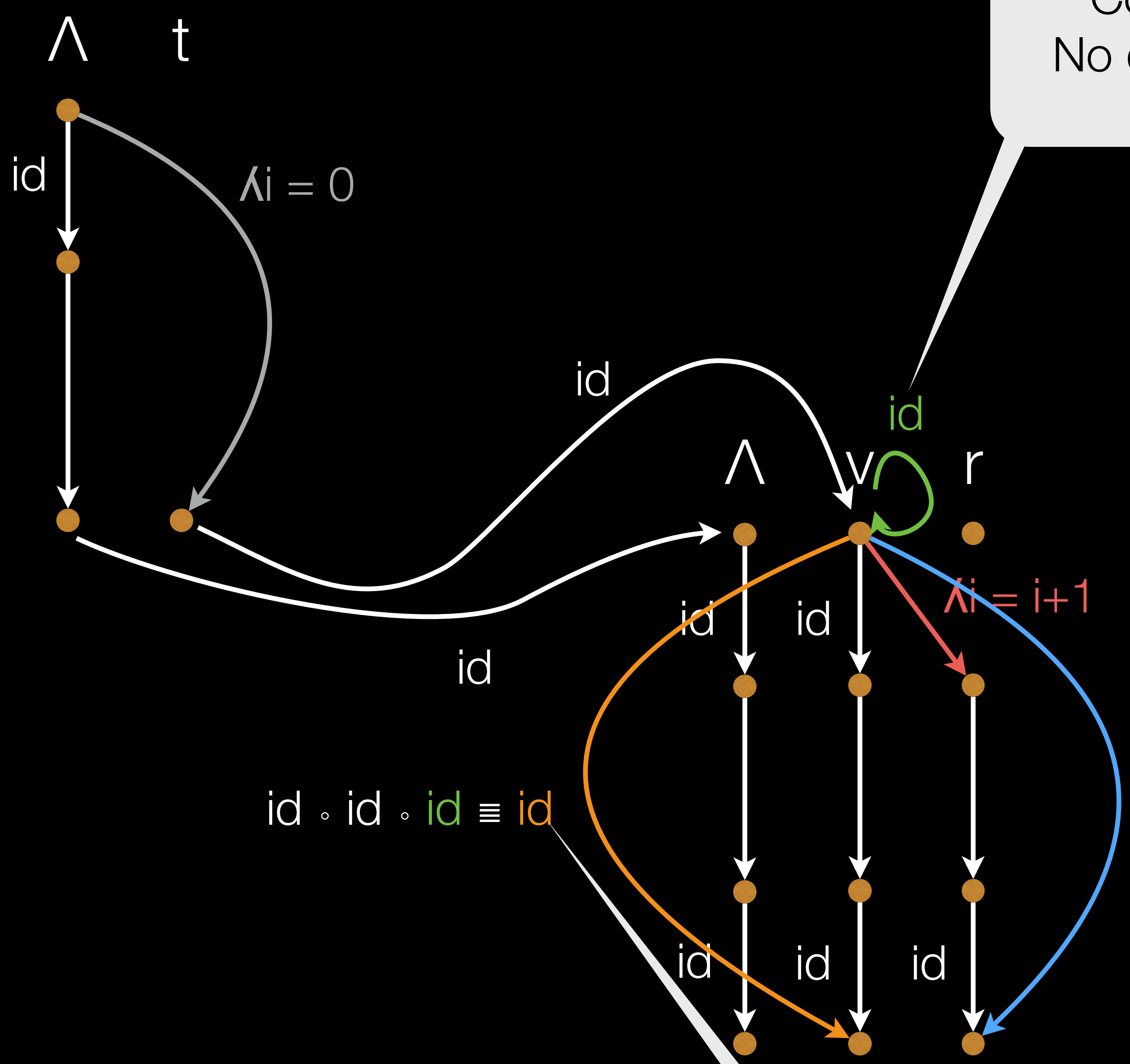
Jump Functions

```
t = 0;
```

```
x = inc(t);
```

```
y = inc(x);
```

```
print(y);
```



Context-independent!
No dependency on λi = 0

id ∘ id ∘ id ≡ id

id ∘ λi = i+1 ∘ id ≡ λi = i+1

```
r = v+1;
```

```
return r;
```

Jump functions to exit points
=> summary functions

@karimhamdanali

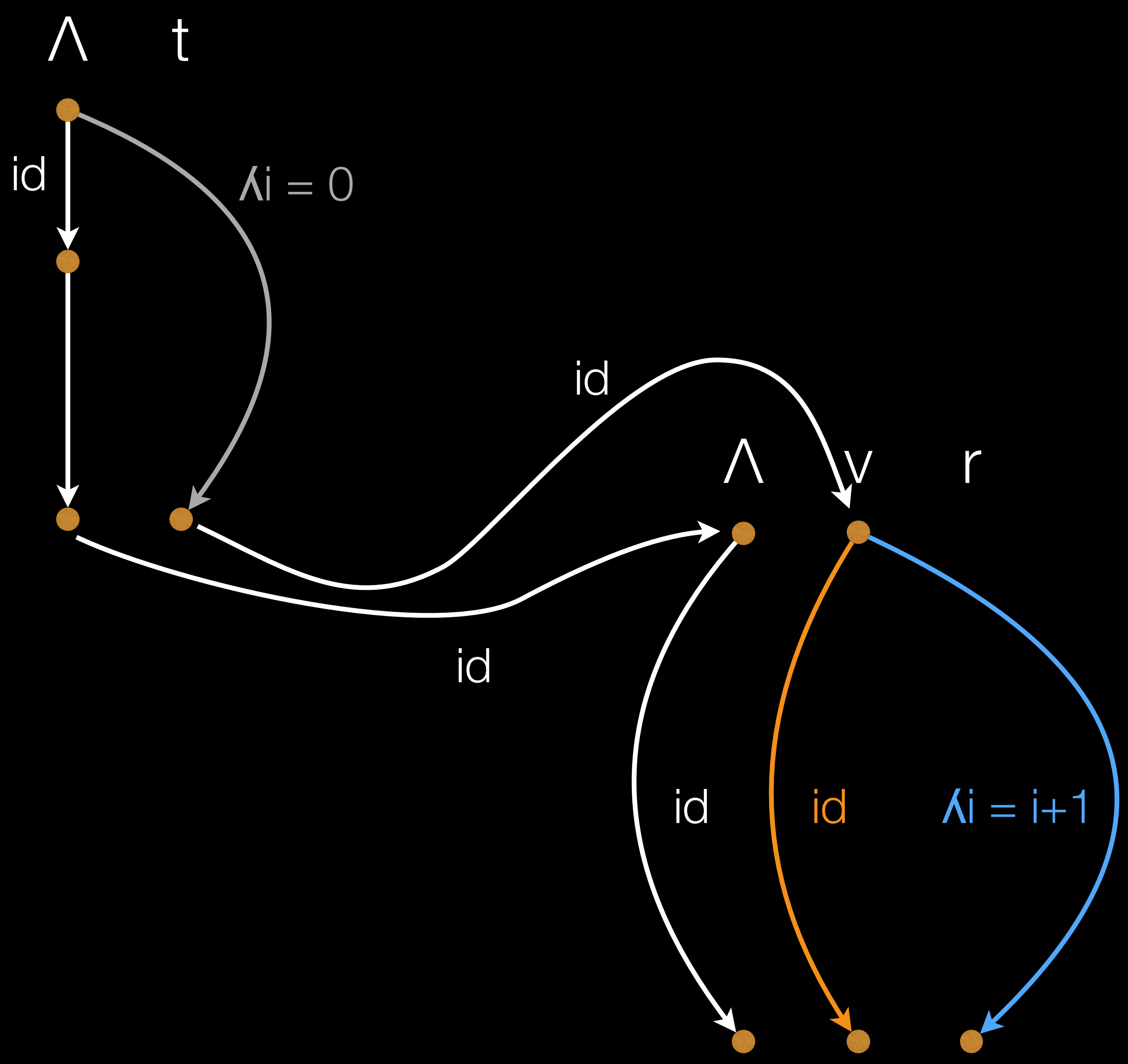
Jump Functions

```
t = 0;
```

```
x = inc(t);
```

```
y = inc(x);
```

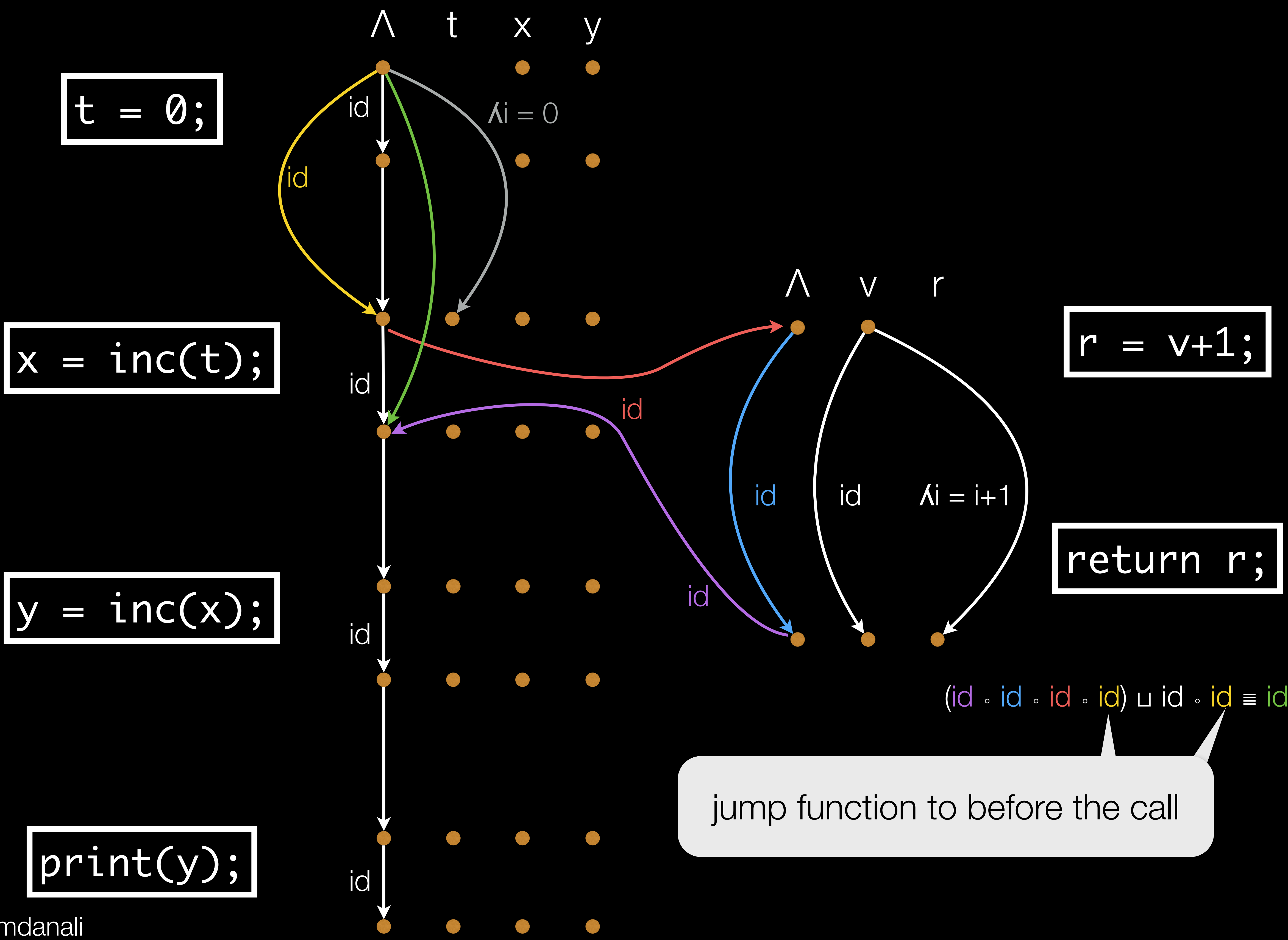
```
print(y);
```



```
r = v+1;
```

```
return r;
```

Jump Functions



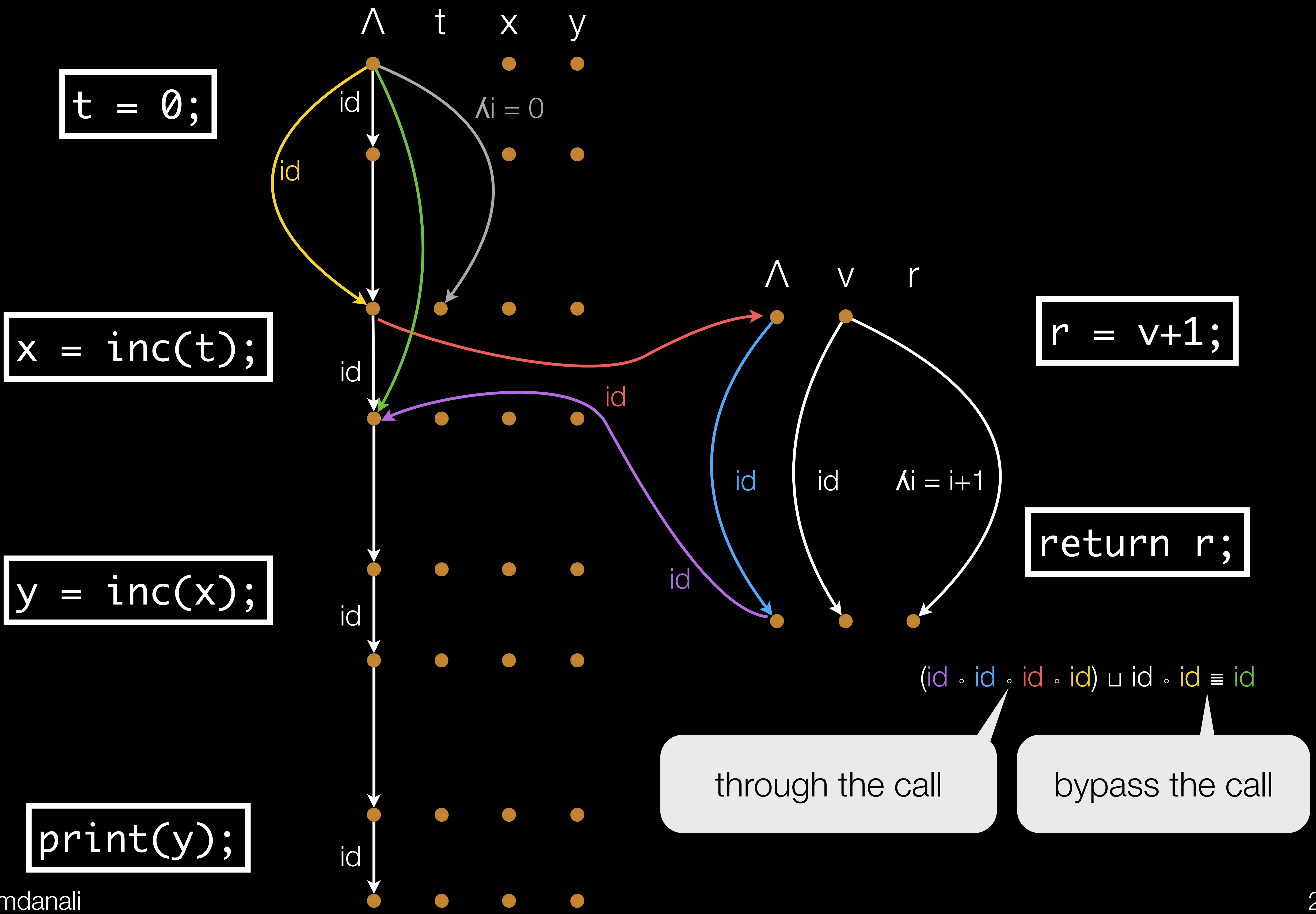
```
t = 0;
x = inc(t);
y = inc(x);
print(y);
```

```
r = v+1;
return r;
```

$(id \circ id \circ id \circ id) \sqcup id \circ id \equiv id$

jump function to before the call

Jump Functions



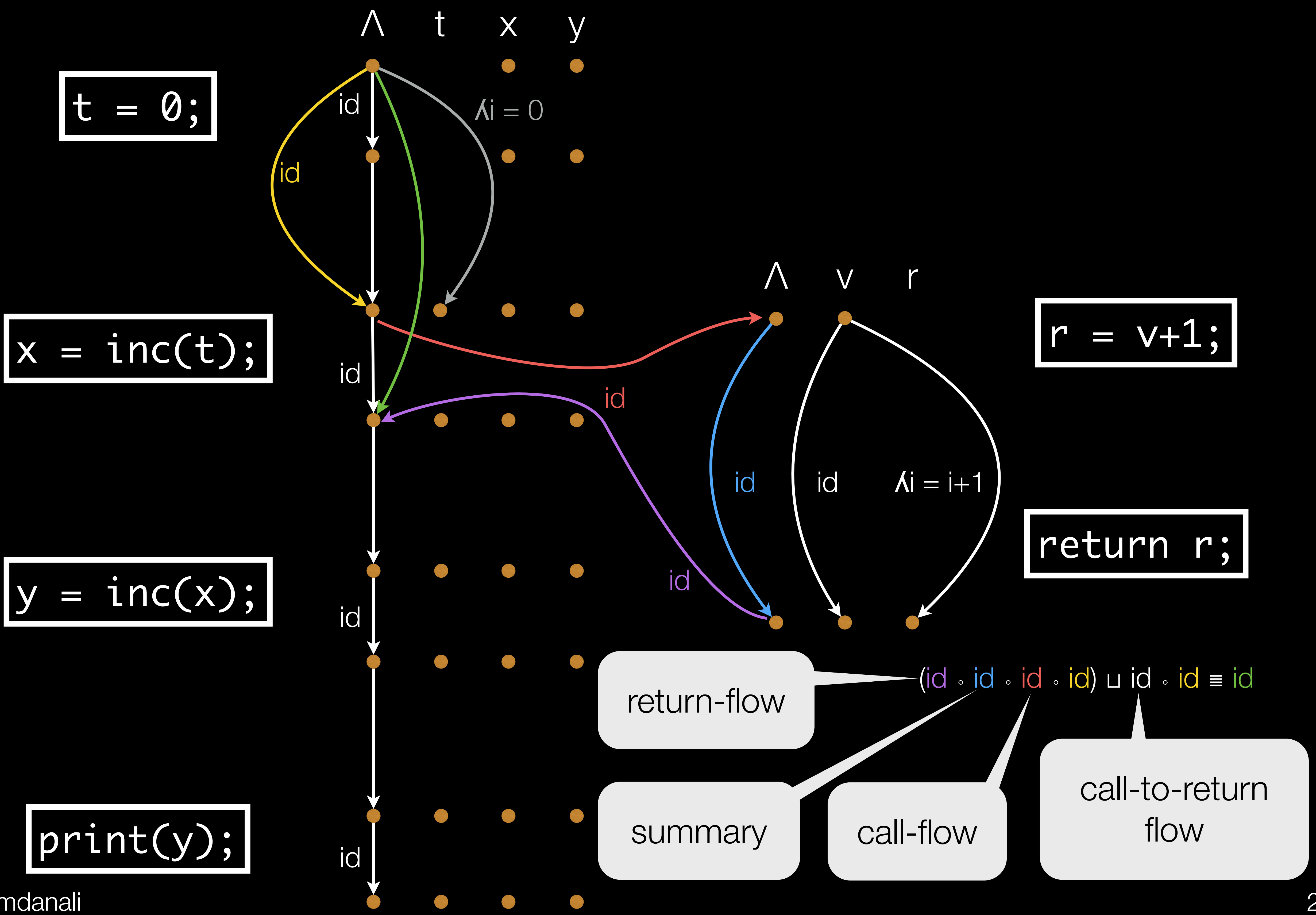
```
t = 0;
x = inc(t);
y = inc(x);
print(y);
r = v+1;
return r;
```

$(\mathrm{id} \circ \mathrm{id} \circ \mathrm{id} \circ \mathrm{id}) \sqcup \mathrm{id} \circ \mathrm{id} \equiv \mathrm{id}$

through the call

bypass the call

Jump Functions



```
t = 0;
x = inc(t);
y = inc(x);
print(y);
```

```
r = v+1;
return r;
```

$(id \circ id \circ id \circ id) \sqcup id \circ id \equiv id$

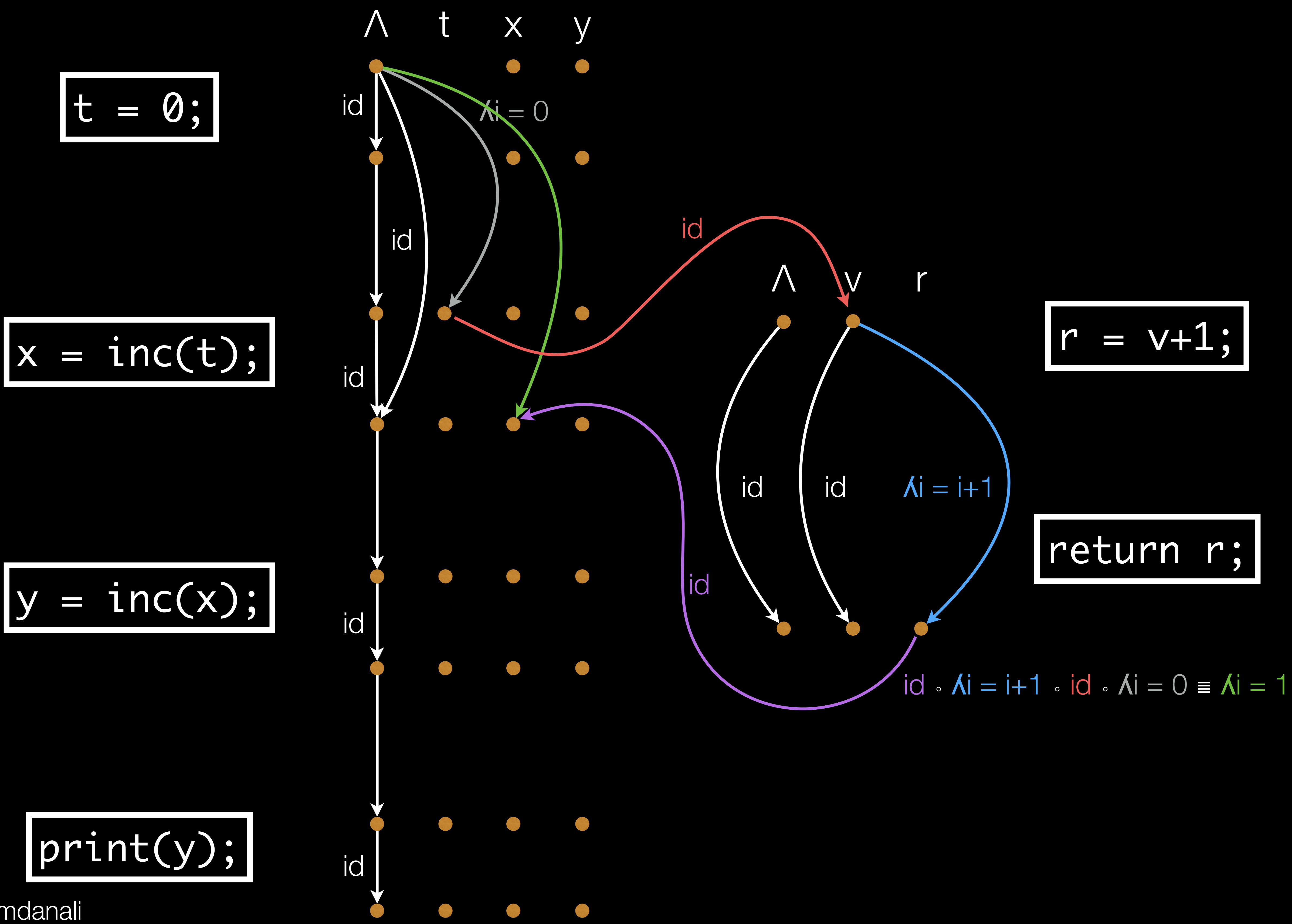
return-flow

summary

call-flow

call-to-return flow

Jump Functions



```
t = 0;
```

```
x = inc(t);
```

```
y = inc(x);
```

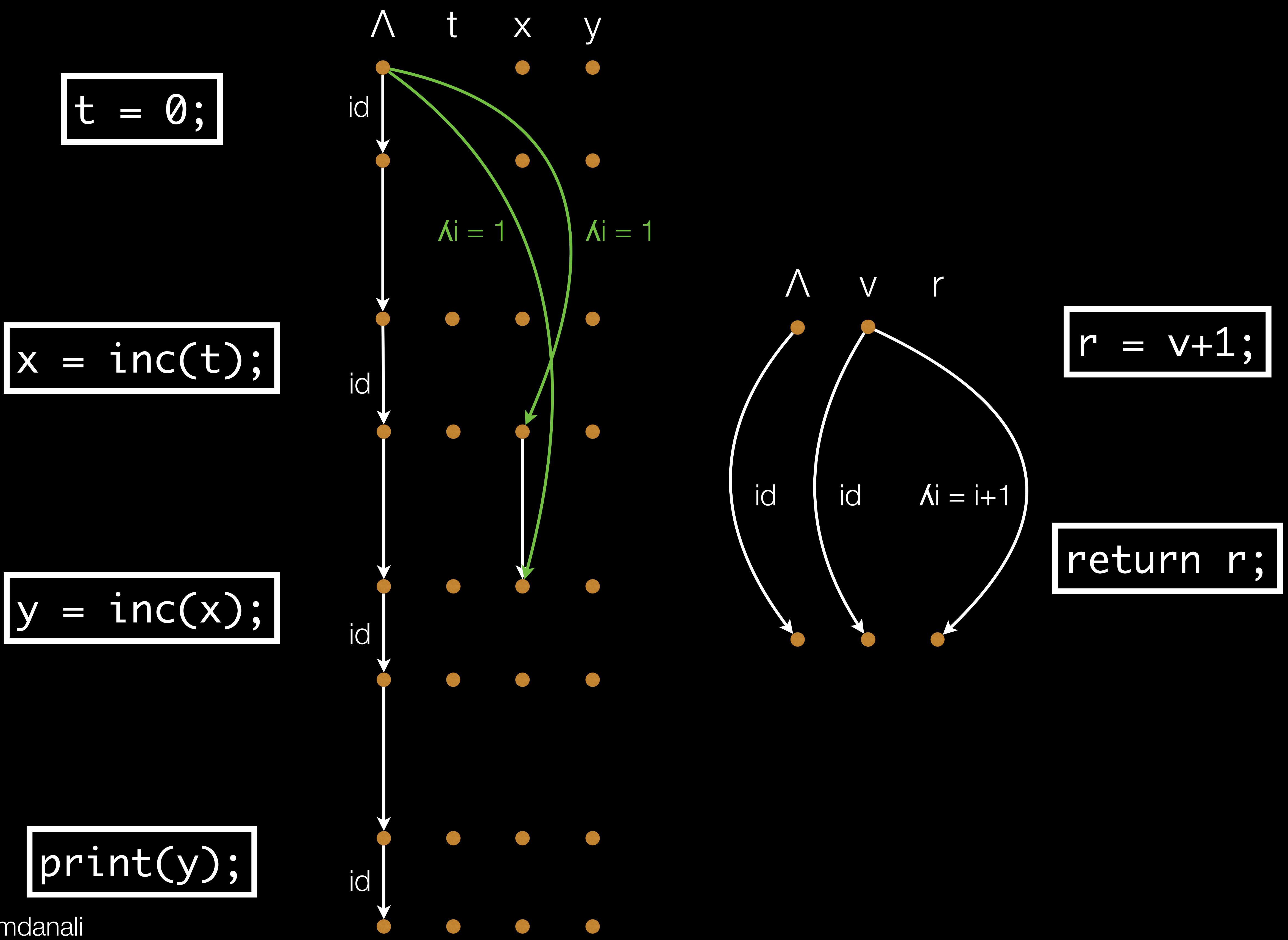
```
print(y);
```

```
r = v+1;
```

```
return r;
```

id ∘ λi = i+1 ∘ id ∘ λi = 0 ≡ λi = 1

Jump Functions



```
t = 0;
x = inc(t);
y = inc(x);
print(y);
```

```
r = v+1;
return r;
```

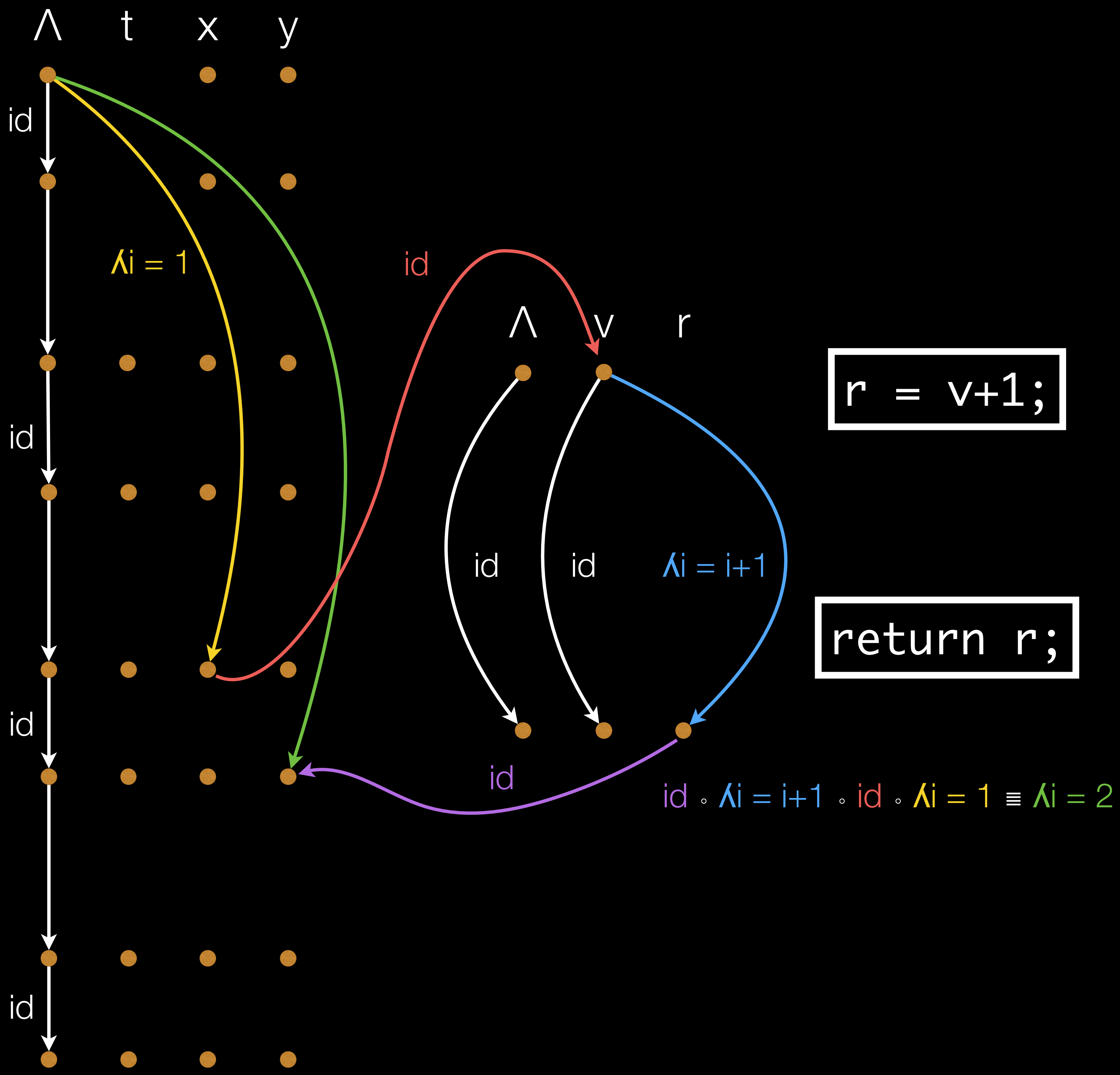
Jump Functions

```
t = 0;
```

```
x = inc(t);
```

```
y = inc(x);
```

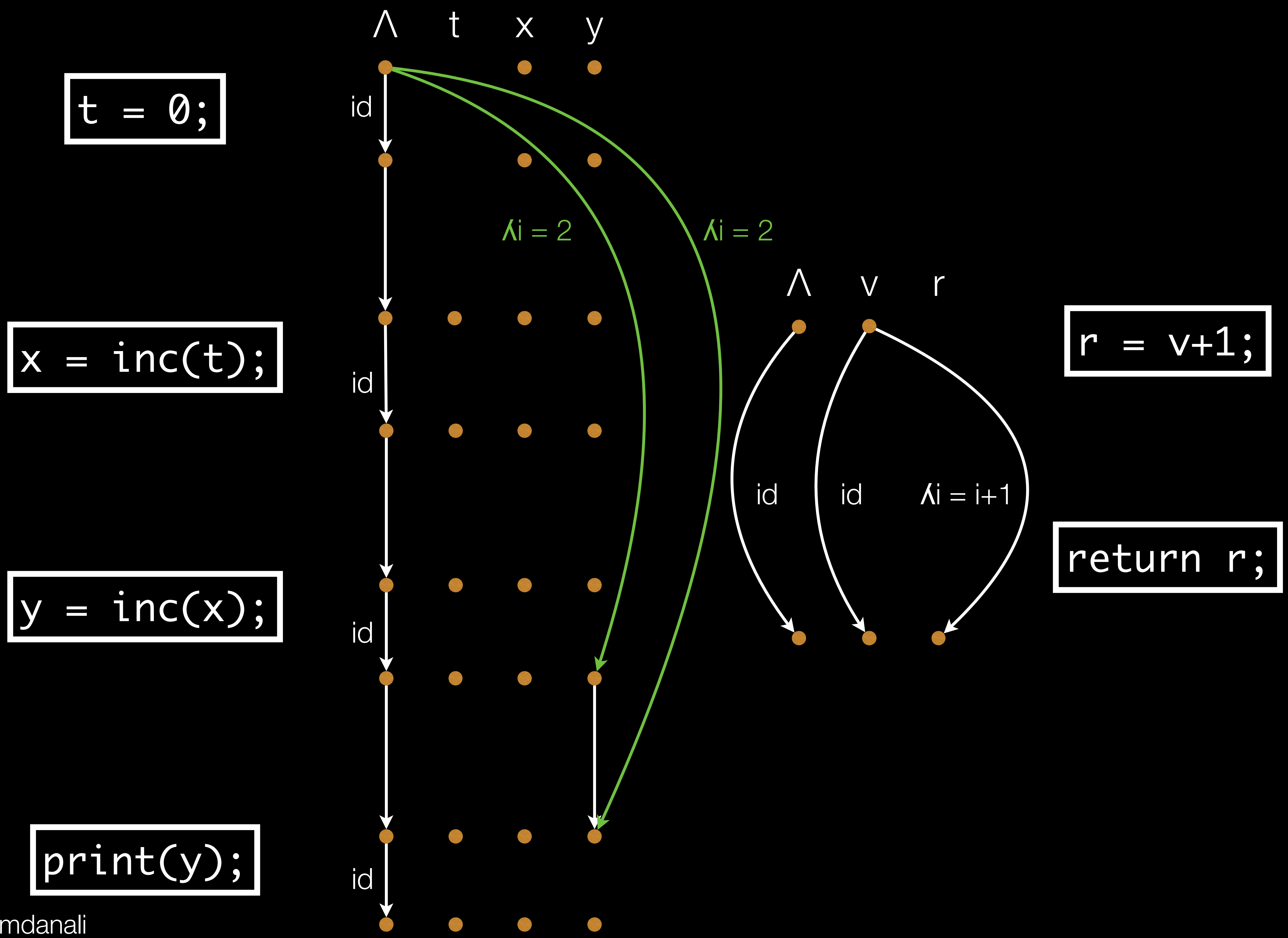
```
print(y);
```



```
r = v+1;
```

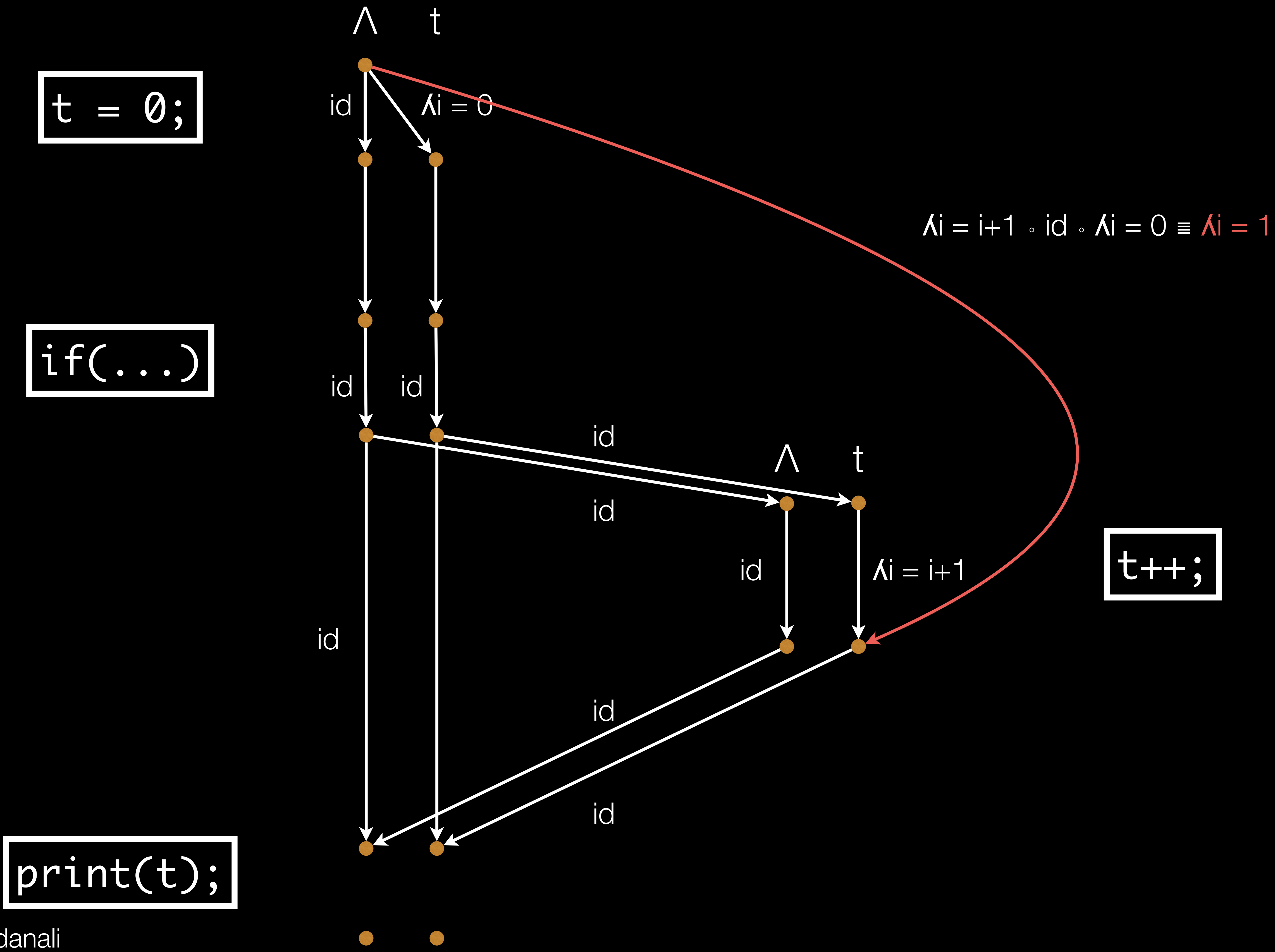
```
return r;
```

Jump Functions



What happens at merge points?

Merge Points



```
t = 0;
```

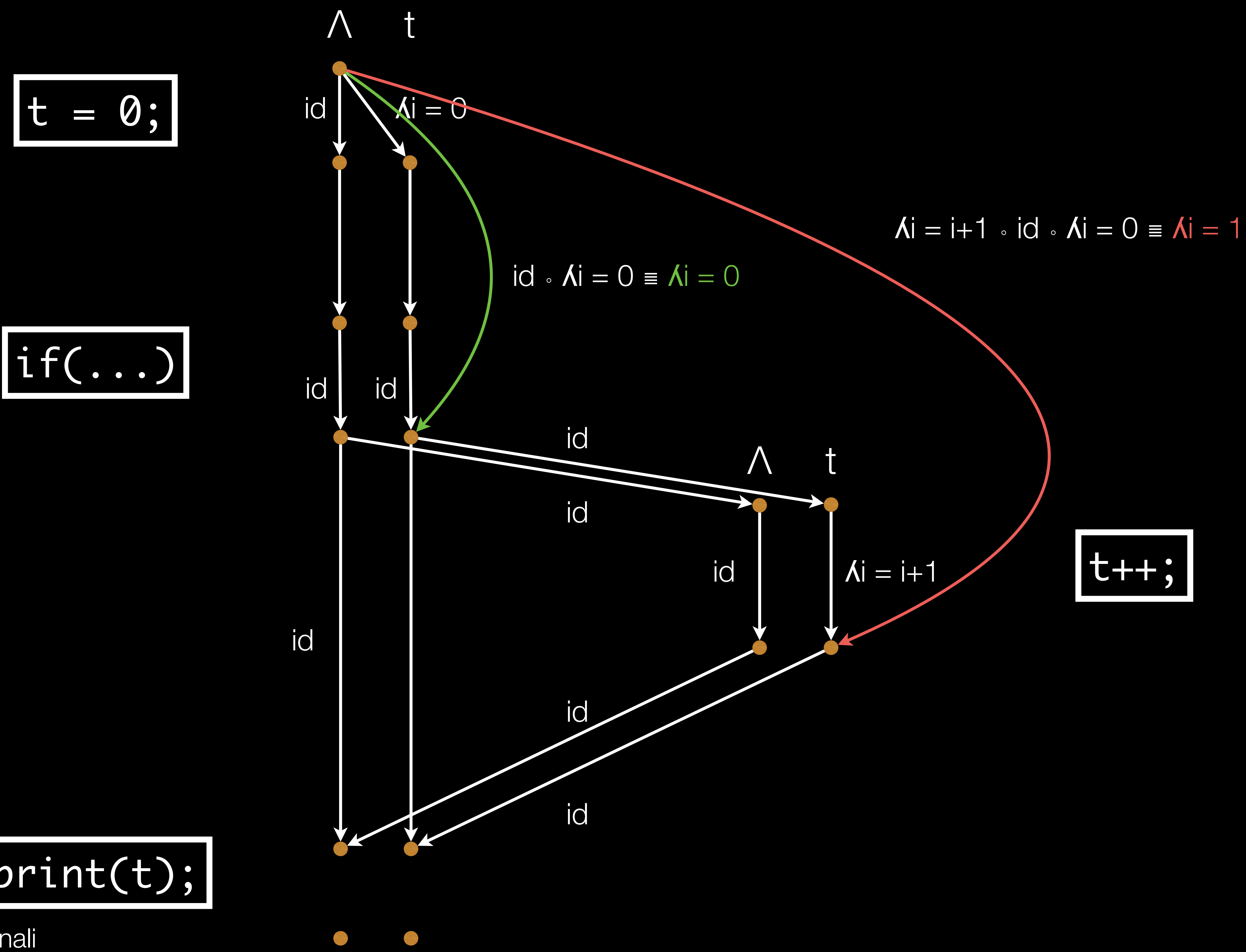
```
if(...)
```

```
t++;
```

```
print(t);
```

$\lambda i = i+1 \circ id \circ \lambda i = 0 \equiv \lambda i = 1$

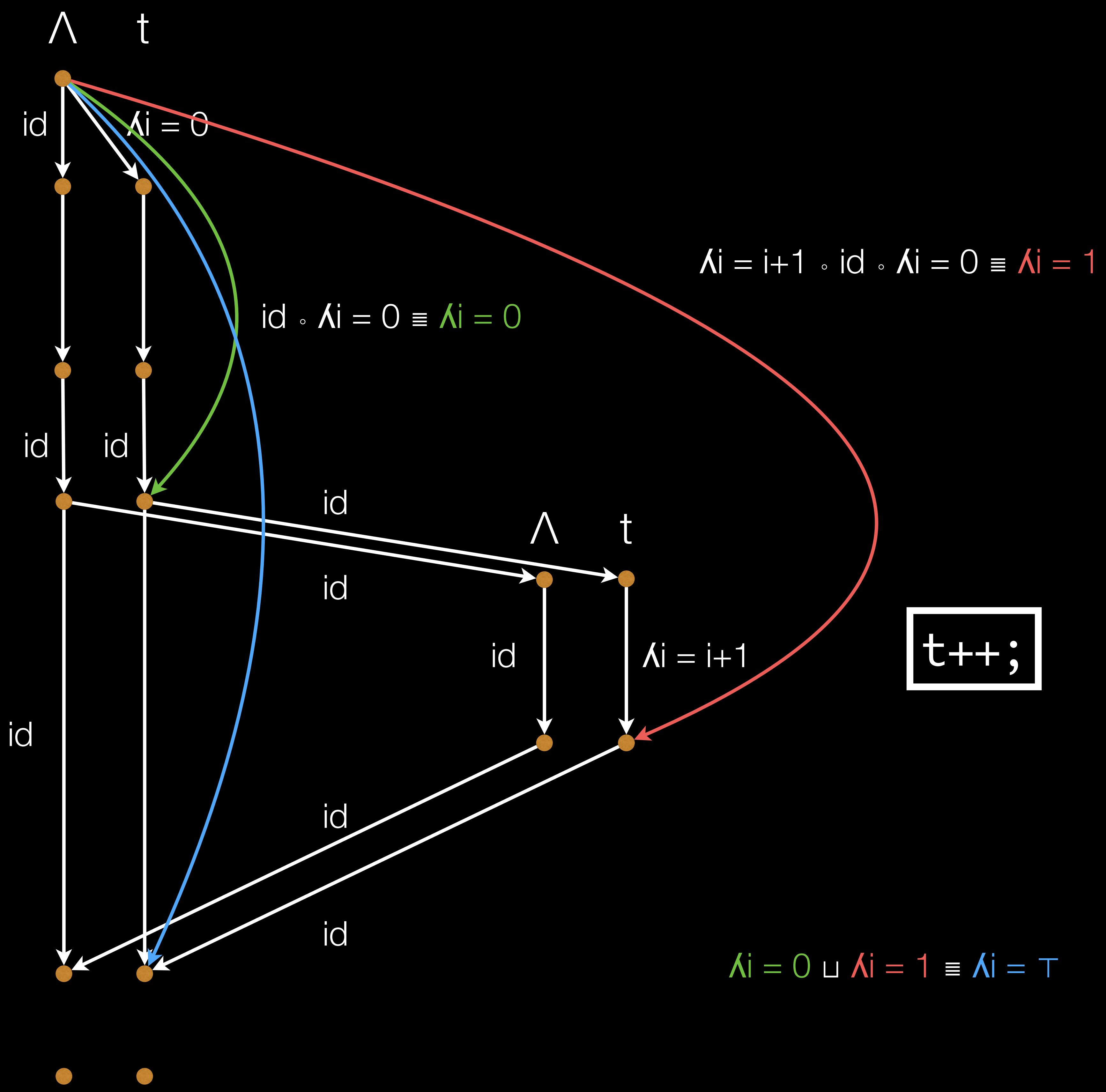
Merge Points



Merge Points

```
t = 0;
```

```
if(...)
```



```
t++;
```

```
print(t);
```

@karimhamdanali

IFDS vs IDE

- IFDS
  - Flow functions
  - Lattice

- IDE
  - Edge functions
  - Merge for edge functions
  - Compose for edge functions

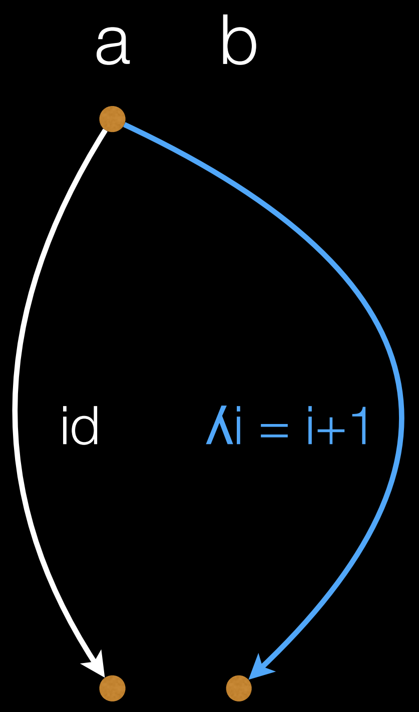
IDE is natural extension of IFDS

- Every IFDS Problem can be encoded as an IDE problem
- Environments map into binary domain $\{\top,\bot\}$
  - effectively represents characteristic function of the flow set
  - $\{a \mapsto \bot, b \mapsto \top, c \mapsto \bot\}$ represents the set {b}
    semantics: b is reachable, a and c are not
  - merge function: $\top \sqcup \top = \top$, $\bot$ otherwise

What are distributive environments?

IDE
Inter-procedural (finite) Distributive Environment problems

- Functions are evaluated over mappings of the kind d↦v ∈ D→L
- Example: {a↦0,b↦2}
- Result is then another environment {a↦0,b↦1}
- Function associated with statement is thus a (distributive) environment transformer



A note on complexity

- Recall IFDS complexity: $O(|E| \cdot |D|^3)$, where D is data-flow facts
- IDE decomposes domain into two domains D and L, where L is lattice values
  - Soot/Heros: the latter is called V, not L
- Exciting result: IDE complexity is also $O(|E| \cdot |D|^3)$
  - independent of size of L, which may be infinite!

Recap

- Computes distributive summary functions, annotated to IFDS data-flow edges
- Callee-side functions are constructed only once but are re-evaluated in every context
    - at every call site, we compose call, summary and return function and then merge with the call-to-return function
    - requires finite representation of function composition and merge

IFDS vs IDE

Feature	IFDS	IDE
Lattice	subset lattice	general lattice
Merge Operator	set union	custom (e.g., min, max, join)
Domain	Facts (D)	Facts (D) + Values (L)
Example Analysis	reaching defs, live variables, taint	constant propagation, interval analysis, taint
Flexibility	limited (Only union facts)	rich (Merge and transform values)

Next

- Synchronized Pushdown Systems (SPDS)