The TAPENADE AD Tool

Laurent Hascoët, Valérie Pascual, Rose-Marie Greborio
Laurent.Hascoet@sophia.inria.fr

Tropics Project, INRIA Sophia-Antipolis

AD Workshop, Cranfield, June 5-6, 2003

PLAN:

- AD: principles of Tangent and Reverse
- Tapenade: technology from Compilation and Parallelization
  - Call Graphs, Flow Graphs, Symbol Tables
  - Static Analyses on Flow Graphs
  - Dependency Analysis
- Tapenade: an AD tool on the web
- Further Developments

AD: Principles of Tangent and Reverse

AD rewrites source programs to make them compute derivatives.

consider: $P: \{I_1; I_2; \ldots I_p;\}$ implementing $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$

AD: Principles of Tangent and Reverse

AD rewrites source programs to make them compute derivatives.

consider:	$P :$	$\{I_1; I_2; \dots I_p;\}$ implementing $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$
identify with:	$f =$	$f_p \circ f_{p-1} \circ \cdots \circ f_1$
name:	$x_0 =$	x and $x_k = f_k(x_{k-1})$

AD: Principles of Tangent and Reverse

AD rewrites source programs to make them compute derivatives.

consider:	$P :$	$\{I_1; I_2; \dots I_p;\}$ implementing $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$
identify with:	$f =$	$f_p \circ f_{p-1} \circ \cdots \circ f_1$
name:	$x_0 =$	x and $x_k = f_k(x_{k-1})$
chain rule:	$f'(x) =$	$f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\dots.f'_1(x_0)$

AD: Principles of Tangent and Reverse

AD rewrites source programs to make them compute derivatives.

consider:	$P:$	$\{I_1; I_2; \dots I_p;\}$ implementing $f : \mathbb{R}^m \to \mathbb{R}^n$
identify with:	$f =$	$f_p \circ f_{p-1} \circ \cdots \circ f_1$
name:	$x_0 =$	x and $x_k = f_k(x_{k-1})$
chain rule:	$f'(x) =$	$f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\dots.f'_1(x_0)$

$f'(x)$ generally too large and expensive $\Rightarrow$ take useful views!

$$\dot{y} = f'(x).\dot{x} = f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\dots.f'_1(x_0).\dot{x} \quad \text{tangent AD}$$

$$\overline{x} = f'^*(x).\overline{y} = f'^*_1(x_0).\dots.f'^*_{p-1}(x_{p-2}).f'^*_p(x_{p-1}).\overline{y} \quad \text{reverse AD}$$

Evaluate both from right to left !

AD: Example

$$\begin{aligned} &\mathbf{\ldots} \\ \mathbf{v_2} &= \mathbf{2} * \mathbf{v_1} + \mathbf{5} \\ \mathbf{v_4} &= \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3} / \mathbf{v_2} \\ &\mathbf{\ldots} \end{aligned}$$

AD: Example

$$\begin{aligned} &\mathbf{\ldots} \\ \mathbf{v_2} &= \mathbf{2} * \mathbf{v_1} + \mathbf{5} \\ \mathbf{v_4} &= \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3} / \mathbf{v_2} \\ &\mathbf{\ldots} \end{aligned}$$

The corresponding (fragment of) Jacobian is:

$$f'(x) = \ldots \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ 0 & 1 - \frac{p_1 * v_3}{v_2^2} & \frac{p_1}{v_2} & 0 \end{pmatrix} \begin{pmatrix} 1 & & & \\ 2 & 0 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \ldots$$

Tangent AD keeps the structure of P:

$$\dot{y} = f'(x).\dot{x} = f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\ldots.f'_1(x_0).\dot{x}$$

$$\begin{aligned} &\mathbf{...} \\ \mathbf{v_2} &= \mathbf{2} * \mathbf{v_1} + \mathbf{5} \\ \mathbf{v_4} &= \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3} / \mathbf{v_2} \\ &\mathbf{...} \end{aligned}$$

Tangent AD keeps the structure of P:

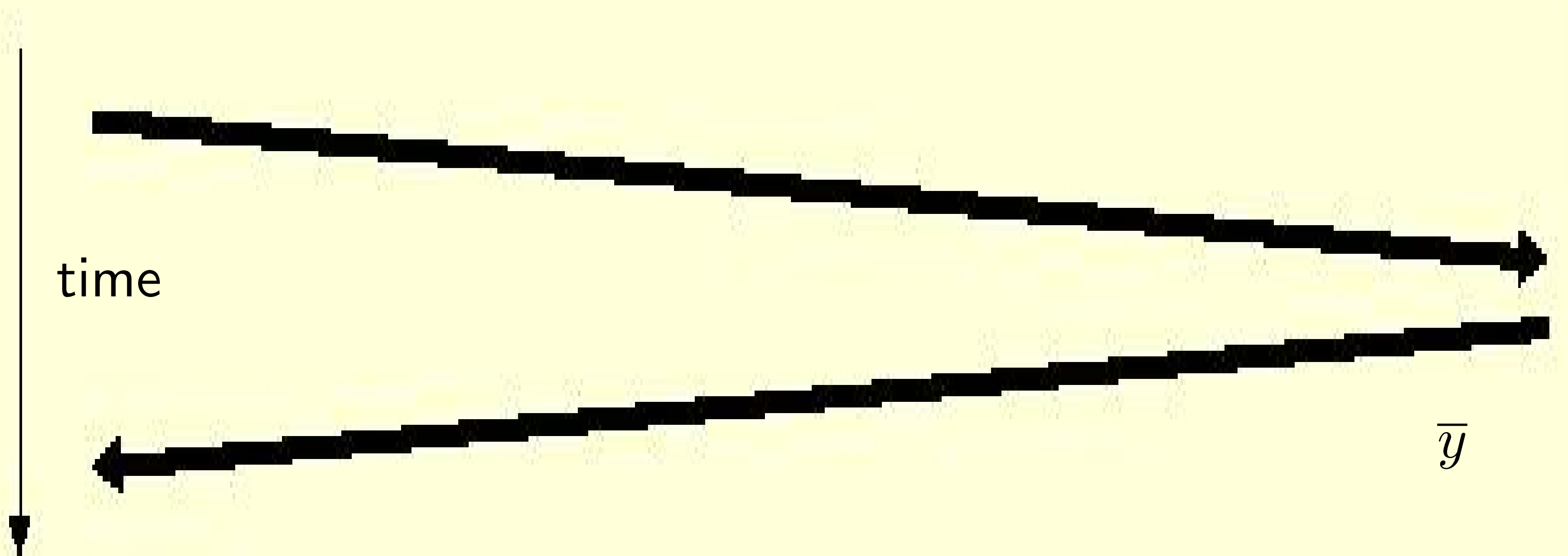
$$\dot{y} = f'(x).\dot{x} = f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\ldots.f'_1(x_0).\dot{x}$$

$$\begin{array}{rcl} & \ldots & \\ \dot{\mathbf{v}}_\mathbf{2} & = & \mathbf{2} * \dot{\mathbf{v}}_\mathbf{1} \\ \mathbf{v}_\mathbf{2} & = & \mathbf{2} * \mathbf{v}_\mathbf{1} \, + \, \mathbf{5} \\ \dot{\mathbf{v}}_\mathbf{4} & = & \dot{\mathbf{v}}_\mathbf{2} * (\mathbf{1} - \mathbf{p}_\mathbf{1} * \mathbf{v}_\mathbf{3}/\mathbf{v}_\mathbf{2}^\mathbf{2}) \, + \, \dot{\mathbf{v}}_\mathbf{3} * \mathbf{p}_\mathbf{1}/\mathbf{v}_\mathbf{2} \\ \mathbf{v}_\mathbf{4} & = & \mathbf{v}_\mathbf{2} \, + \, \mathbf{p}_\mathbf{1} * \mathbf{v}_\mathbf{3}/\mathbf{v}_\mathbf{2} \\ & \ldots & \end{array}$$

just inserts the products $\dot{x}_k = f'_k(x_{k-1})$ for $k = 1$ to p.

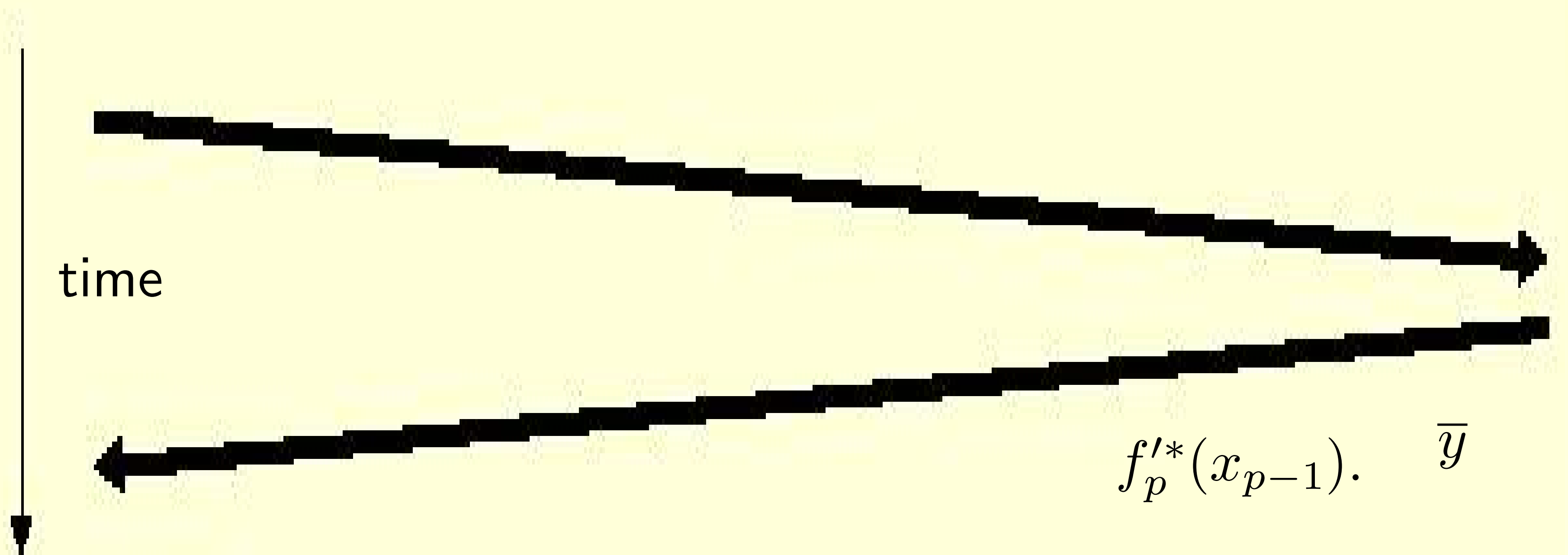
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^{*}(x).\overline{y} = f_1'^{*}(x_0).\ldots.f_{p-1}'^{*}(x_{p-2}).f_p'^{*}(x_{p-1}).\overline{y}$$



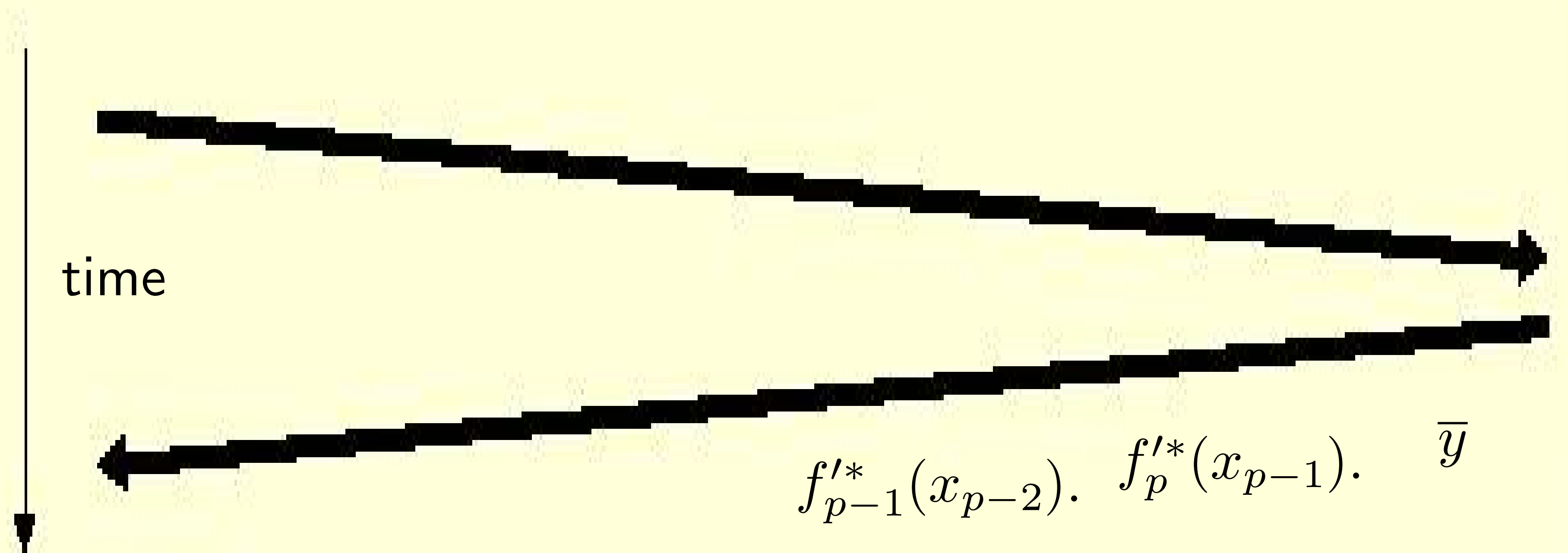
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^*(x).\overline{y} = f_1'^*(x_0).\ldots.f_{p-1}'^*(x_{p-2}).f_p'^*(x_{p-1}).\overline{y}$$



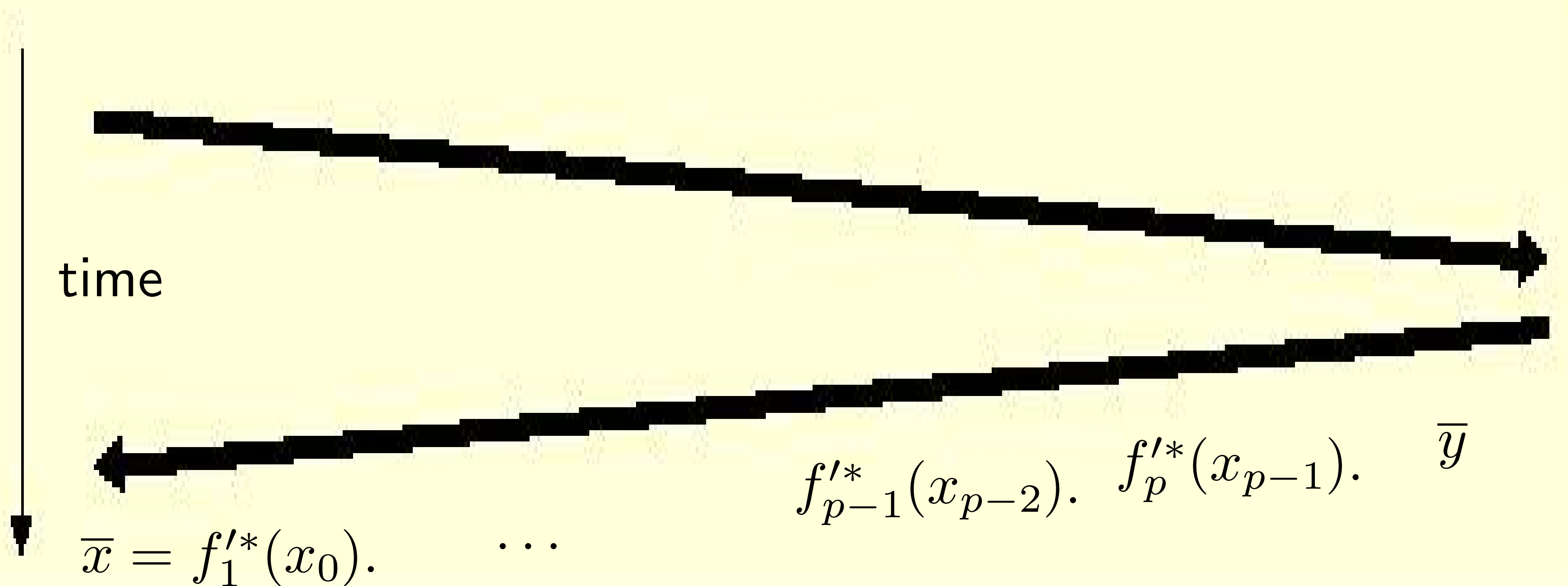
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^*(x).\overline{y} = f_1'^*(x_0).\ldots.f_{p-1}'^*(x_{p-2}).f_p'^*(x_{p-1}).\overline{y}$$



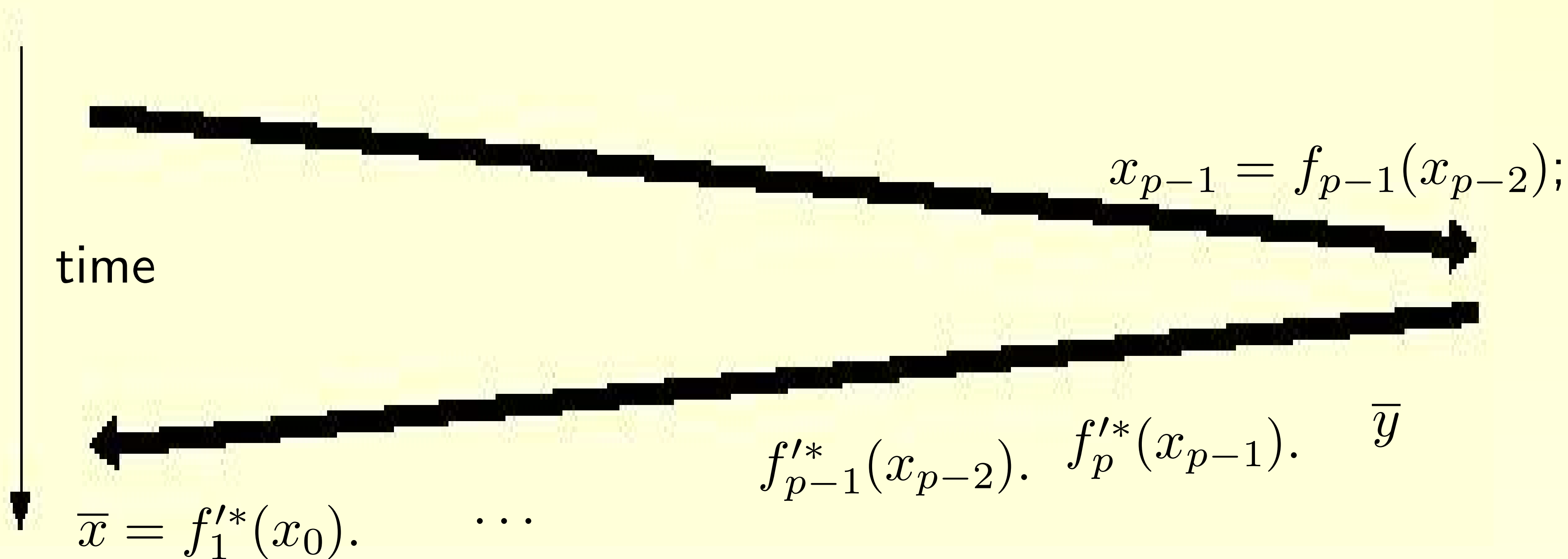
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^{*}(x).\overline{y} = f_1'^{*}(x_0). \ldots . f_{p-1}'^{*}(x_{p-2}).f_p'^{*}(x_{p-1}).\overline{y}$$



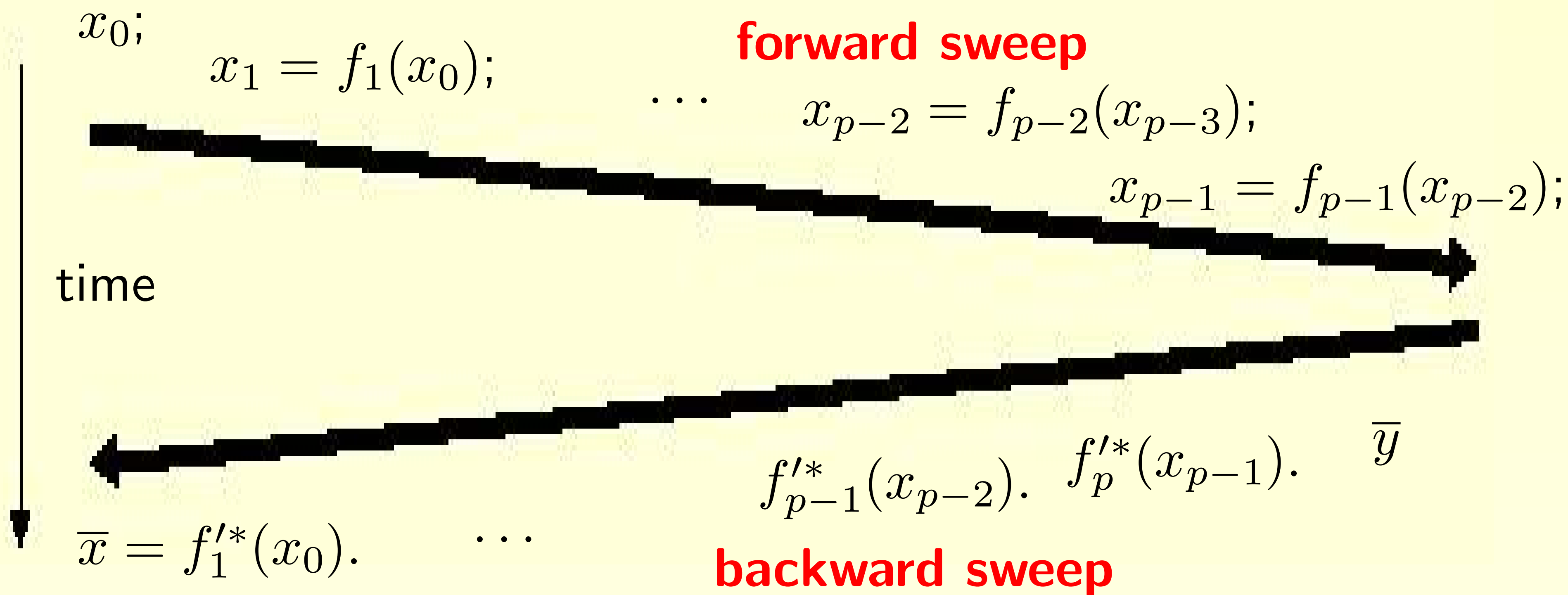
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^*(x).\overline{y} = f_1'^*(x_0). \ldots . f_{p-1}'^*(x_{p-2}).f_p'^*(x_{p-1}).\overline{y}$$



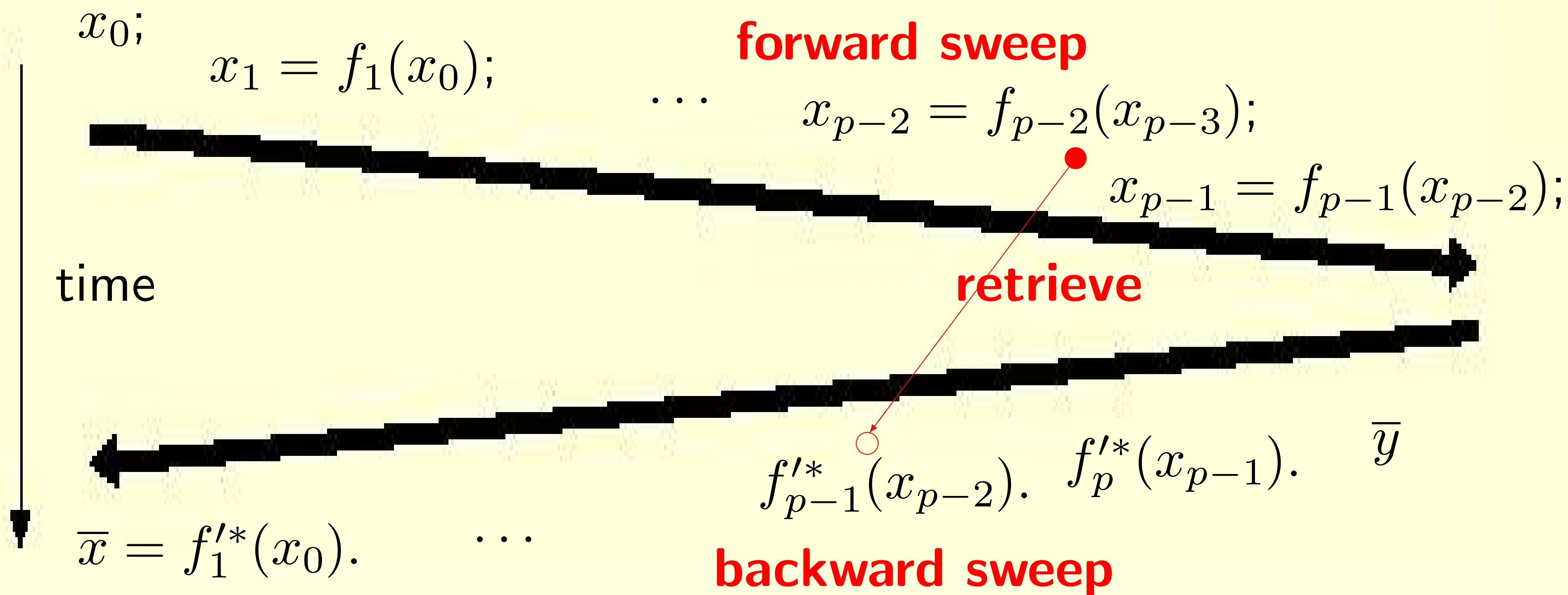
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^{*}(x).\overline{y} = f_1'^{*}(x_0).\ldots.f_{p-1}'^{*}(x_{p-2}).f_p'^{*}(x_{p-1}).\overline{y}$$



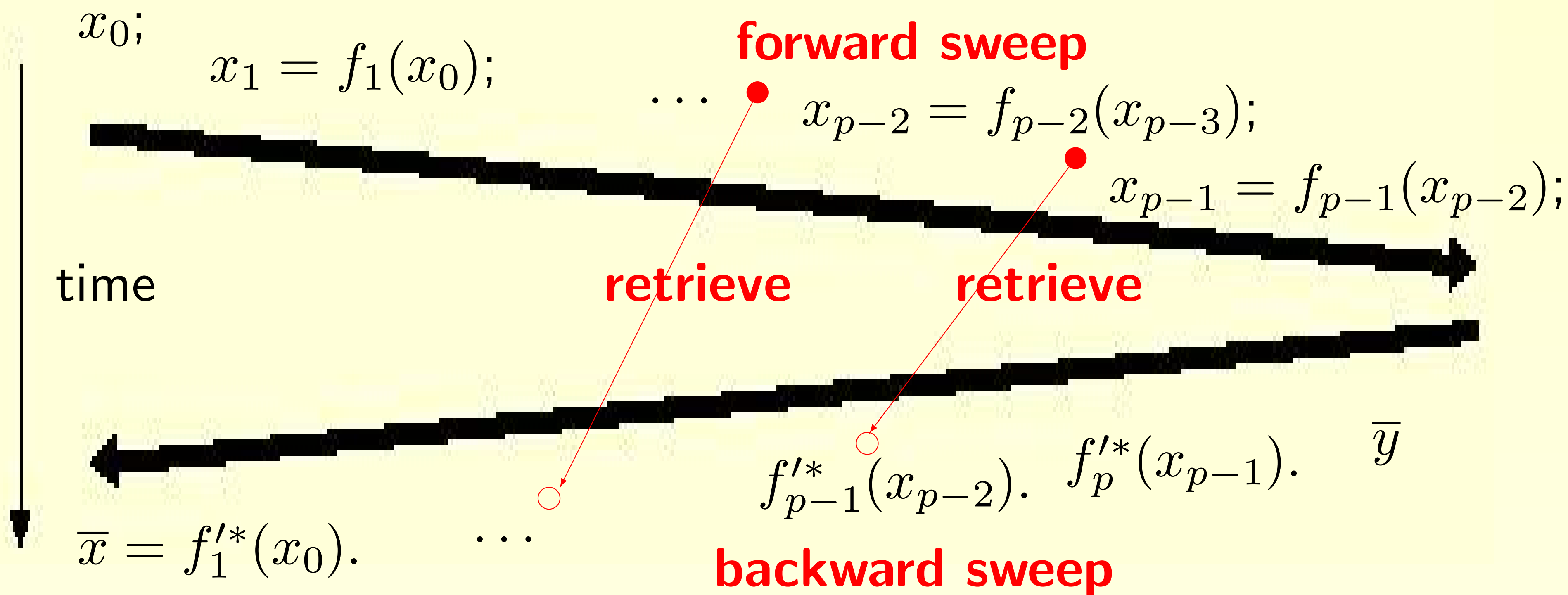
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^*(x).\overline{y} = f_1'^*(x_0). \ldots f_{p-1}'^*(x_{p-2}).f_p'^*(x_{p-1}).\overline{y}$$



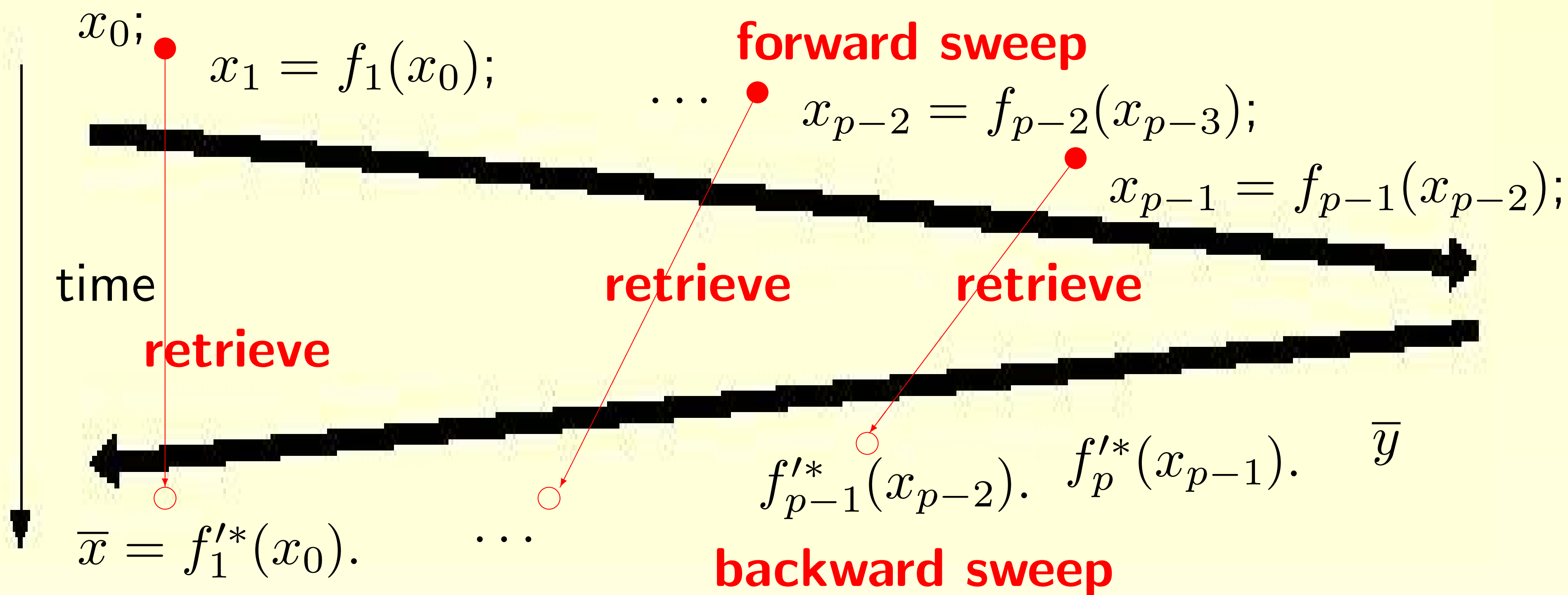
AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^{*}(x).\overline{y} = f_1'^{*}(x_0)\ldots.f_{p-1}'^{*}(x_{p-2}).f_p'^{*}(x_{p-1}).\overline{y}$$



AD: Reverse is more tricky than Tangent

$$\overline{x} = f'^{*}(x).\overline{y} = f_1'^{*}(x_0).\ldots.f_{p-1}'^{*}(x_{p-2}).f_p'^{*}(x_{p-1}).\overline{y}$$



Memory usage ("Tape") is the bottleneck!

AD: Continued Example

Program fragment:

$$\begin{aligned} & \ldots \\ \mathbf{v_2} &= \mathbf{2} * \mathbf{v_1} + \mathbf{5} \\ \mathbf{v_4} &= \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3} / \mathbf{v_2} \\ & \ldots \end{aligned}$$

AD: Continued Example

Program fragment:

$$
\begin{aligned}
& \ldots \\
\mathbf{v_2} &= \mathbf{2} * \mathbf{v_1} + \mathbf{5} \\
\mathbf{v_4} &= \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3} / \mathbf{v_2} \\
& \ldots
\end{aligned}
$$

Corresponding transposed Partial Jacobians:

$$
f'^*(x) = \ldots \begin{pmatrix} 1 & 2 & & \\ & 0 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & 0 \\ & 1 & & 1 - \frac{p_1 * v_3}{v_2^2} \\ & & 1 & \frac{p_1}{v_2} \\ & & & 0 \end{pmatrix} \ldots
$$

AD: Reverse mode on the example

$$\ldots$$

$$\begin{aligned}
\bar{\mathbf{v}}_2 &= \bar{\mathbf{v}}_2 + \bar{\mathbf{v}}_4 * (\mathbf{1} - \mathbf{p}_1 * \mathbf{v}_3/\mathbf{v}_2^2) \\
\bar{\mathbf{v}}_3 &= \bar{\mathbf{v}}_3 + \bar{\mathbf{v}}_4 * \mathbf{p}_1/\mathbf{v}_2 \\
\bar{\mathbf{v}}_4 &= \mathbf{0}
\end{aligned}$$

$$\begin{aligned}
\bar{\mathbf{v}}_1 &= \bar{\mathbf{v}}_1 + \mathbf{2} * \bar{\mathbf{v}}_2 \\
\bar{\mathbf{v}}_2 &= \mathbf{0}
\end{aligned}$$

$$\ldots$$

AD: Reverse mode on the example

$$\ldots$$

$$\mathbf{v_2} = \mathbf{2} * \mathbf{v_1} + \mathbf{5}$$

$$\mathbf{v_4} = \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3}/\mathbf{v_2}$$

$$\ldots$$

$$\ldots$$

$$\begin{aligned} \bar{\mathbf{v}}_2 &= \bar{\mathbf{v}}_2 + \bar{\mathbf{v}}_4 * (\mathbf{1} - \mathbf{p_1} * \mathbf{v_3}/\mathbf{v_2^2}) \\ \bar{\mathbf{v}}_3 &= \bar{\mathbf{v}}_3 + \bar{\mathbf{v}}_4 * \mathbf{p_1}/\mathbf{v_2} \\ \bar{\mathbf{v}}_4 &= \mathbf{0} \end{aligned}$$

$$\begin{aligned} \bar{\mathbf{v}}_1 &= \bar{\mathbf{v}}_1 + \mathbf{2} * \bar{\mathbf{v}}_2 \\ \bar{\mathbf{v}}_2 &= \mathbf{0} \end{aligned}$$

$$\ldots$$

AD: Reverse mode on the example

$$\ldots$$

Push(v_2)

$$\mathbf{v_2} = \mathbf{2} * \mathbf{v_1} + \mathbf{5}$$

Push(v_4)

$$\mathbf{v_4} = \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3}/\mathbf{v_2}$$

$$\ldots$$

$$\ldots$$

Pop(v_4)

$$\bar{\mathbf{v}}_\mathbf{2} = \bar{\mathbf{v}}_\mathbf{2} + \bar{\mathbf{v}}_\mathbf{4} * (\mathbf{1} - \mathbf{p_1} * \mathbf{v_3}/\mathbf{v_2^2})$$

$$\bar{\mathbf{v}}_\mathbf{3} = \bar{\mathbf{v}}_\mathbf{3} + \bar{\mathbf{v}}_\mathbf{4} * \mathbf{p_1}/\mathbf{v_2}$$

$$\bar{\mathbf{v}}_\mathbf{4} = \mathbf{0}$$

Pop(v_2)

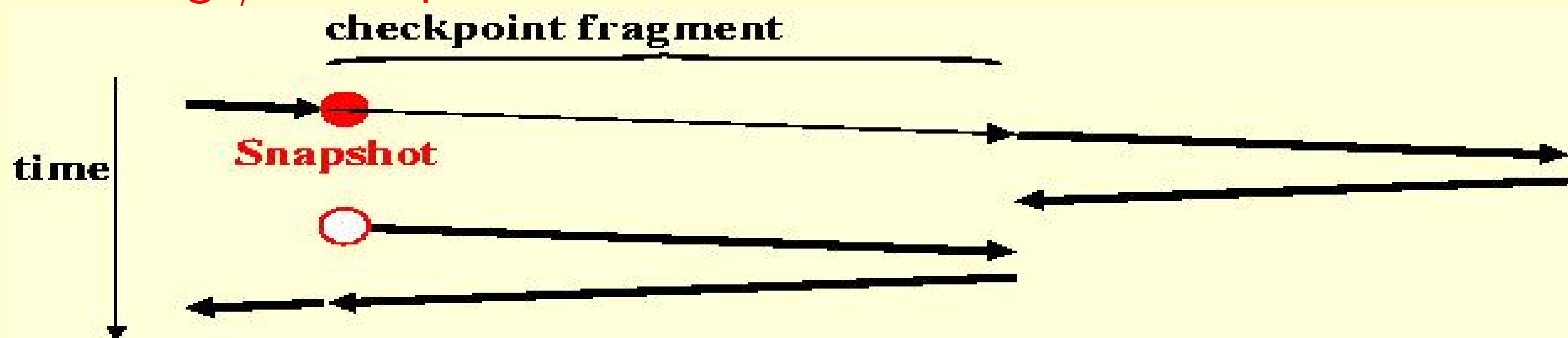
$$\bar{\mathbf{v}}_\mathbf{1} = \bar{\mathbf{v}}_\mathbf{1} + \mathbf{2} * \bar{\mathbf{v}}_\mathbf{2}$$

$$\bar{\mathbf{v}}_\mathbf{2} = \mathbf{0}$$

$$\ldots$$

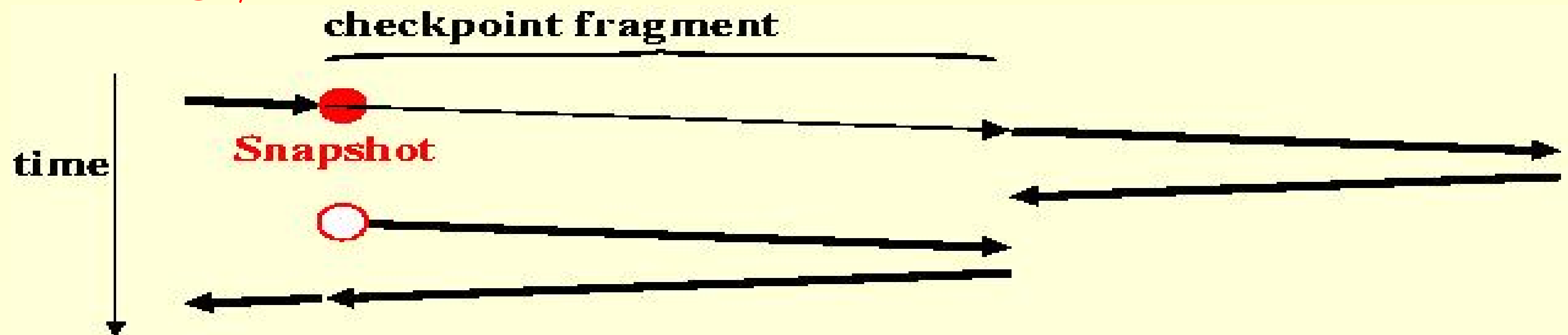
AD: The Checkpointing tactic

A Storage/Recomputation tradeoff:

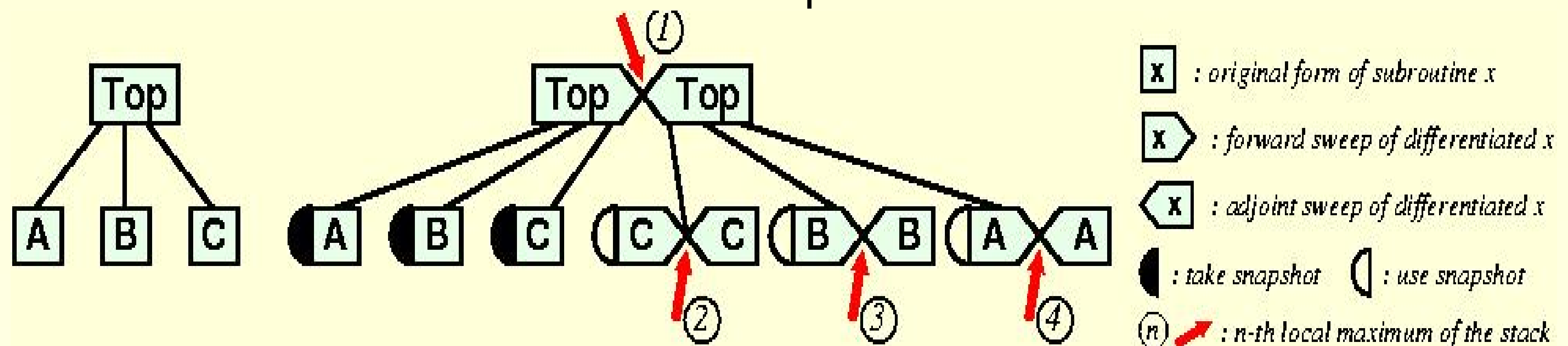


AD: The Checkpointing tactic

A Storage/Recomputation tradeoff:



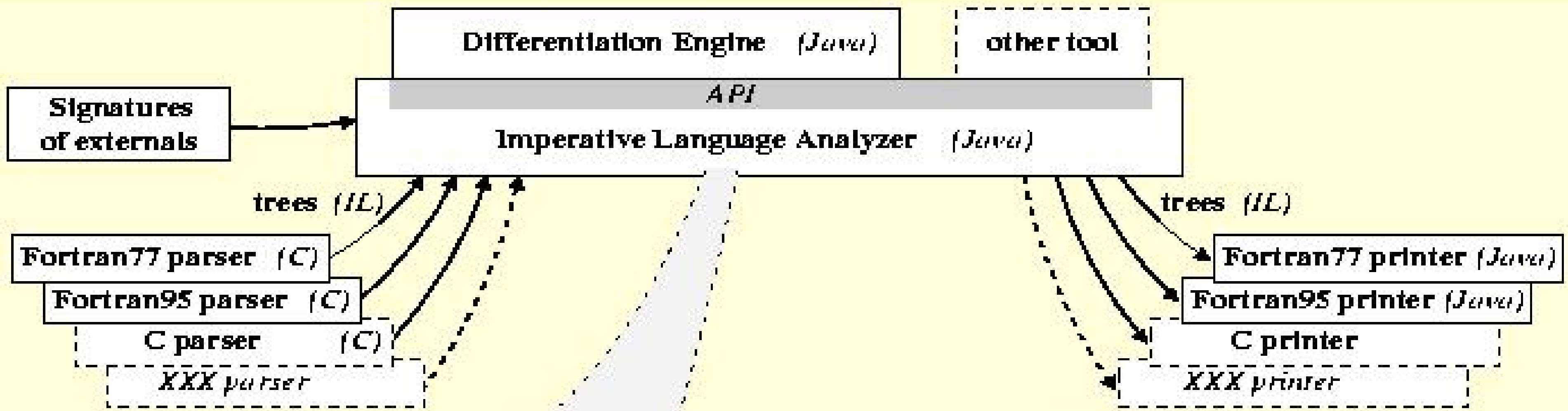
TAPENADE does it on the Call Graph :



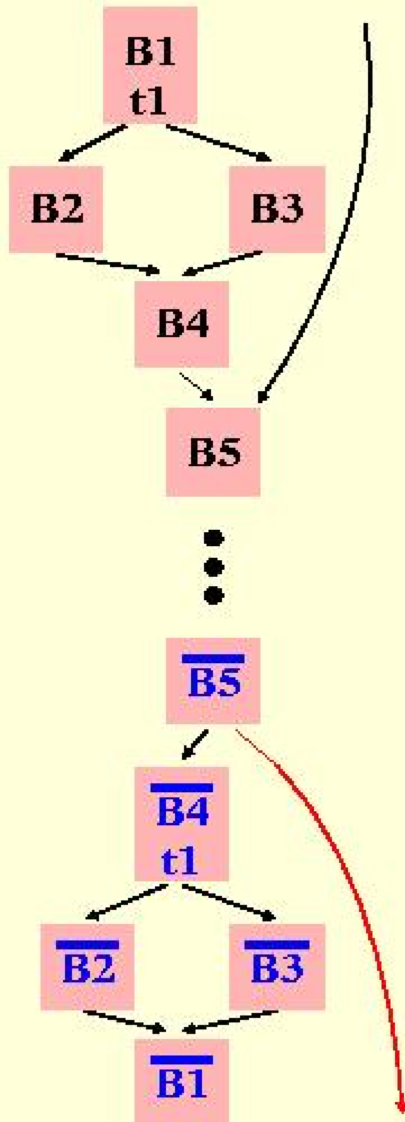
Tapenade: Internal Representation

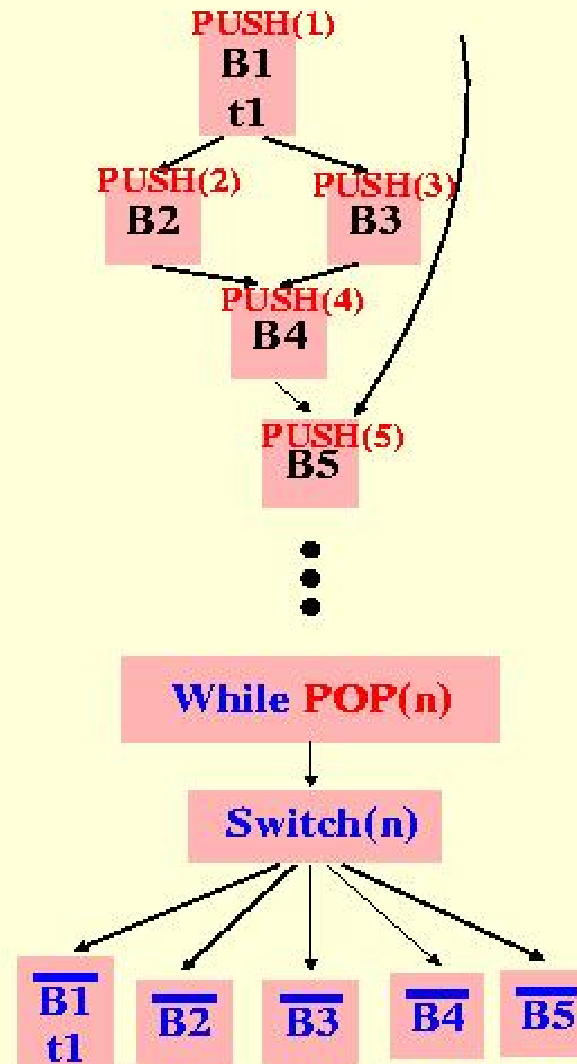
Take profit of well-known techniques
from Compilation and Parallelization:

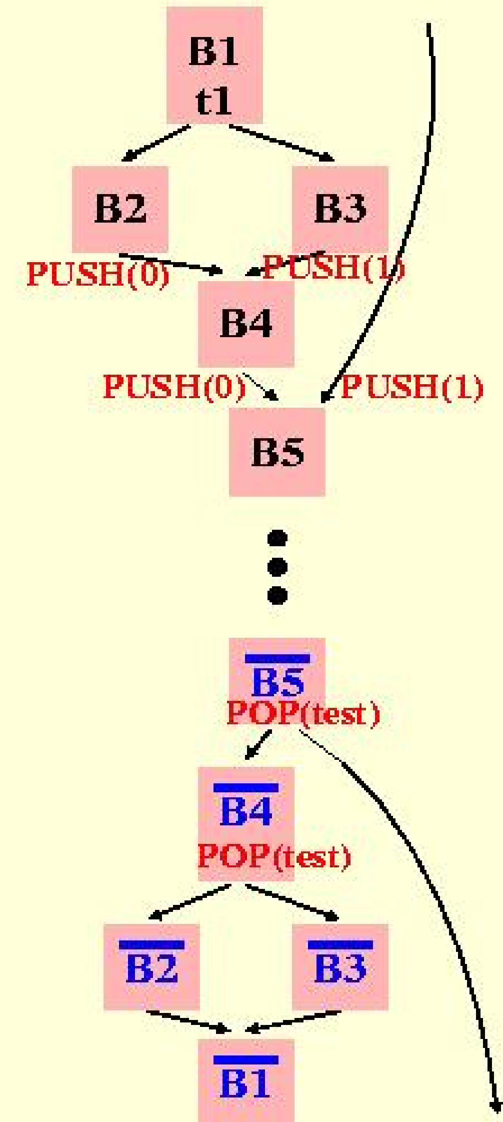
- Use a general abstract *Imperative Language (IL)*
- Represent programs as *Call Graphs* of *Flow Graphs*
- Store symbol declarations in nested *Symbol Tables*


Differentiation Engine (Java)
other tool
API
Signatures of externals
Imperative Language Analyzer (Java)
trees (IL)
Fortran77 parser (C)
Fortran95 parser (C)
C parser (C)
XXX parser
trees (IL)
Fortran77 printer (Java)
Fortran95 printer (Java)
C printer
XXX printer
Top
A
B
C
Entry
small(A,B)
n = 0
1,100
push
Header
DO 100 i=1,100
do exit
do loop
pop
cycle
IF (A(i).ge.0)
if true
if false
pop
A(i) = n
n = n + 1
B(i) = 0
A(i) = B(i)
print *,n
Exit
end

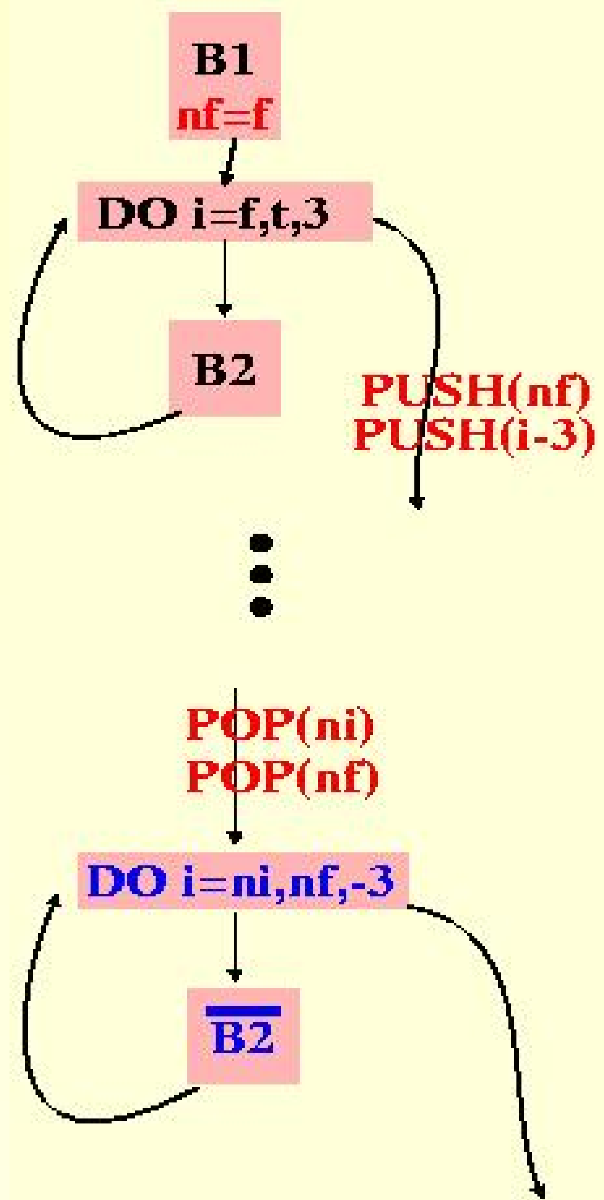
Application: Inversion of the Flow Graph







Application: Loop Inversion

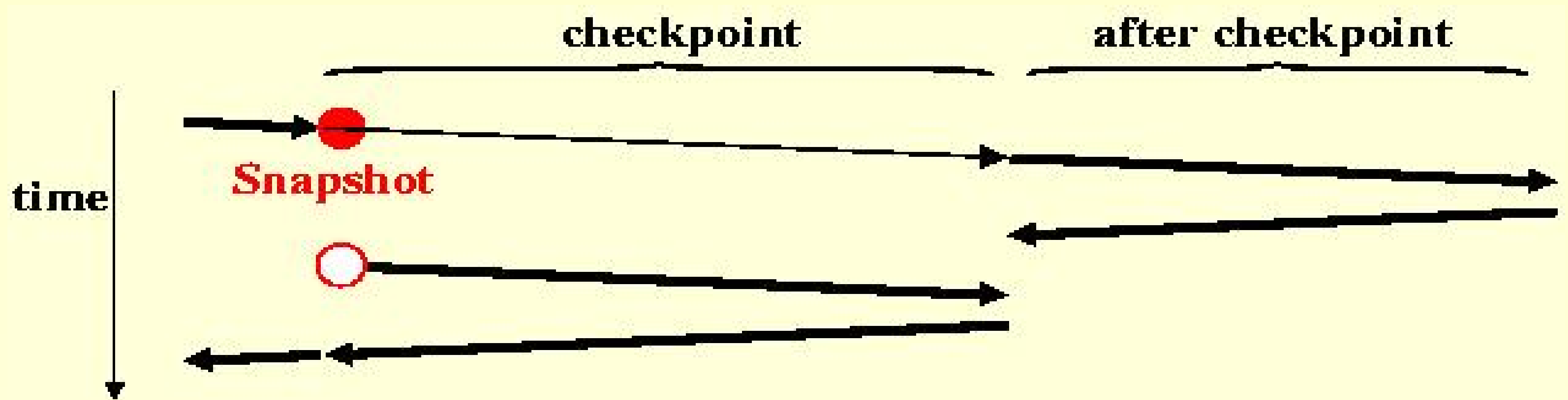


Tapenade: Global Static Analyses on Flow Graphs

- classical *IN-OUT* analysis.
- forward dependence with respect to *independent* inputs.
- backward influence on *dependent* inputs.
- specific *TBR* analysis for the reverse mode.
- . . . *pointer* analysis . . .

Usual restrictions: conservative assumptions, arrays . . .

Application: reduced snapshots



Snapshot = **IN**(**checkpoint**) $\bigcap$ **OUT**(**checkpoint and after**)

Tapenade: Using Data Dependencies

flow: *write x*$\longrightarrow$*read x*

anti: *read x*$\longrightarrow$*write x*

output: *write x*$\longrightarrow$*write x*

Data Dependencies form

- a partial order between run-time instructions.
- a graph between textual instructions.

Any instructions shuffle that respects Data Dependencies is valid !

Application: Loop Fusion in "Vector" Mode

...

$$\mathbf{a} = \mathbf{2.0} * \mathbf{a} + \mathbf{10.0}$$

$$\mathbf{b} = \mathbf{c} + \mathbf{sin}(\mathbf{a})$$

$$\mathbf{c} = \mathbf{0.0}$$

...

Application: Loop Fusion in "Vector" Mode

$$
\begin{aligned}
&\quad \mathbf{\ldots} \\
&\mathbf{Do\ \ n = 1, ndt} \\
&\quad \dot{\mathbf{a}}(\mathbf{n}) \ = \ \mathbf{2.0} * \dot{\mathbf{a}}(\mathbf{n}) \\
&\mathbf{Enddo} \\
&\mathbf{a} \ = \ \mathbf{2.0} * \mathbf{a} \ + \ \mathbf{10.0} \\
&\mathbf{Do\ \ n = 1, ndt} \\
&\quad \dot{\mathbf{b}}(\mathbf{n}) \ = \ \dot{\mathbf{c}}(\mathbf{n}) + \mathbf{cos}(\mathbf{a}) * \dot{\mathbf{a}}(\mathbf{n}) \\
&\mathbf{Enddo} \\
&\mathbf{b} \ = \ \mathbf{c} \ + \ \mathbf{sin}(\mathbf{a}) \\
&\mathbf{Do\ \ n = 1, ndt} \\
&\quad \dot{\mathbf{c}}(\mathbf{n}) \ = \ \mathbf{0.0} \\
&\mathbf{Enddo} \\
&\mathbf{c} \ = \ \mathbf{0.0} \\
&\quad \mathbf{\ldots}
\end{aligned}
$$

Application: Loop Fusion in "Vector" Mode

$$
\begin{aligned}
&\quad\ldots\\
&\mathbf{Do}\ \ \mathbf{n = 1, ndt}\\
&\quad \dot{\mathbf{a}}(\mathbf{n}) \ = \ \mathbf{2.0} * \dot{\mathbf{a}}(\mathbf{n})\\
&\mathbf{Enddo}\\
&\mathbf{a} \ = \ \mathbf{2.0} * \mathbf{a} \ + \ \mathbf{10.0}\\
&\mathbf{Do}\ \ \mathbf{n = 1, ndt}\\
&\quad \dot{\mathbf{b}}(\mathbf{n}) \ = \ \dot{\mathbf{c}}(\mathbf{n}) + \mathbf{cos}(\mathbf{a}) * \dot{\mathbf{a}}(\mathbf{n})\\
&\mathbf{Enddo}\\
&\mathbf{b} \ = \ \mathbf{c} \ + \ \mathbf{sin}(\mathbf{a})\\
&\mathbf{Do}\ \ \mathbf{n = 1, ndt}\\
&\quad \dot{\mathbf{c}}(\mathbf{n}) \ = \ \mathbf{0.0}\\
&\mathbf{Enddo}\\
&\mathbf{c} \ = \ \mathbf{0.0}\\
&\quad\ldots
\end{aligned}
$$

$$
\begin{aligned}
&\quad\ldots\\
&\mathbf{a} \ = \ \mathbf{2.0} * \mathbf{a} \ + \ \mathbf{10.0}\\
&\mathbf{Do}\ \ \mathbf{n = 1, ndt}\\
&\quad \dot{\mathbf{a}}(\mathbf{n}) \ = \ \mathbf{2.0} * \dot{\mathbf{a}}(\mathbf{n})\\
&\quad \dot{\mathbf{b}}(\mathbf{n}) \ = \ \dot{\mathbf{c}}(\mathbf{n}) + \mathbf{cos}(\mathbf{a}) * \dot{\mathbf{a}}(\mathbf{n})\\
&\quad \dot{\mathbf{c}}(\mathbf{n}) \ = \ \mathbf{0.0}\\
&\mathbf{Enddo}\\
&\mathbf{b} \ = \ \mathbf{c} \ + \ \mathbf{sin}(\mathbf{a})\\
&\mathbf{c} \ = \ \mathbf{0.0}\\
&\quad\ldots
\end{aligned}
$$

Tapenade: an AD tool on the web

Differentiation result - Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward Reload Stop http://tapenade.inria.fr:8080/tapenade/result.html Search Print

Home Bookmarks Internet Lookup New&Cool

Retry with the same files

Original call graph

```
adj
  sub2
  sub1
  maxx
```

Download differentiated file

Differentiated call graph

```
adj_dv
  maxx_dv
  sub1_dv
  sub2_dv
```

```
C
      SUBROUTINE ADJ(u, z, t)
      REAL t, u, z
      REAL x(14), y
      COMMON /cc/ x, y
      INTEGER i, MAXX
      REAL v
      EXTERNAL MAXX
C
      i = 5
      x(1) = y * u + t
      z = MAXX(z, t)
      u = 0.0
      CALL SUB1(u, x(i), z, v)
      t = t + x(1) * z + 3 * v
      y = 0.0
      i = 6
      CALL SUB2(u, x(3), z, v)
```

```
      x(1) = y * u + t
      CALL MAXX_DV(z, zd, t, td, z)
      u = 0.0
      CALL SUB1_DV(u, ud, x(i), xd(1
      DO nd=1,nbdirs
        td(nd) = td(nd) + z * xd(nd,
      ENDDO
      t = t + x(1) * z + 3 * v
      y = 0.0
      i = 6
      CALL SUB2_DV(u, ud, x(3), xd(1
      DO nd=1,nbdirs
        td(nd) = td(nd) + z * xd(nd,
      ENDDO
      t = t + x(1) * z + 3 * u
      DO nd=1,nbdirs
        zd(nd) = 0.0
      ENDDO
```

```
2 adj: undeclared external routine: maxx
3 adj: Return type of maxx set by implicit rule to INTEGER
4 adj: argument type mismatch in call of sub1, REAL(0:6) expected, receives I
5 adj: argument type mismatch in call of sub2, REAL(0:12) expected, receives
6 maxx: Tool: Please provide a differentiated function for unit maxx for argu
```

Document: Done (0.11 secs)

- Servlet on http://www-sop.inria.fr/tropics or batch
- Uploads your Files and Includes
- Displays results and messages with links to source

Future work...

Tapenade now 18 months old.
Several applications: Aeronautics, Hydrology, Chemistry, Biology...
Many developments still waiting:

- User Directives: active I-O, checkpoints, special loops
- `FORTRAN95`, and then `C`
- Dead code in the Reverse mode
- Validity domain for derivatives