TAPENADE: a tool for Automatic Differentiation of programs

Laurent Hascoët
Laurent.Hascoet@sophia.inria.fr

TROPICS Project, INRIA Sophia-Antipolis

ECCOMAS 2004, Jyväskylä, July 25-28, 2004

PLAN:

- AD: principles of Tangent and Reverse
- Tapenade: technology from Compilation and Parallelization
- Tapenade: differentiation model on examples
- Tapenade: an AD tool on the web
- Conclusion and 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:	$f'(x) =$	$f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\ldots.f'_1(x_0)$

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

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

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

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

Evaluate both from right to left !

AD: Example

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

AD: Example

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

The corresponding (fragment of) Jacobian is:

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

Tangent AD keeps the structure of P:

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

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

Tangent AD keeps the structure of P:

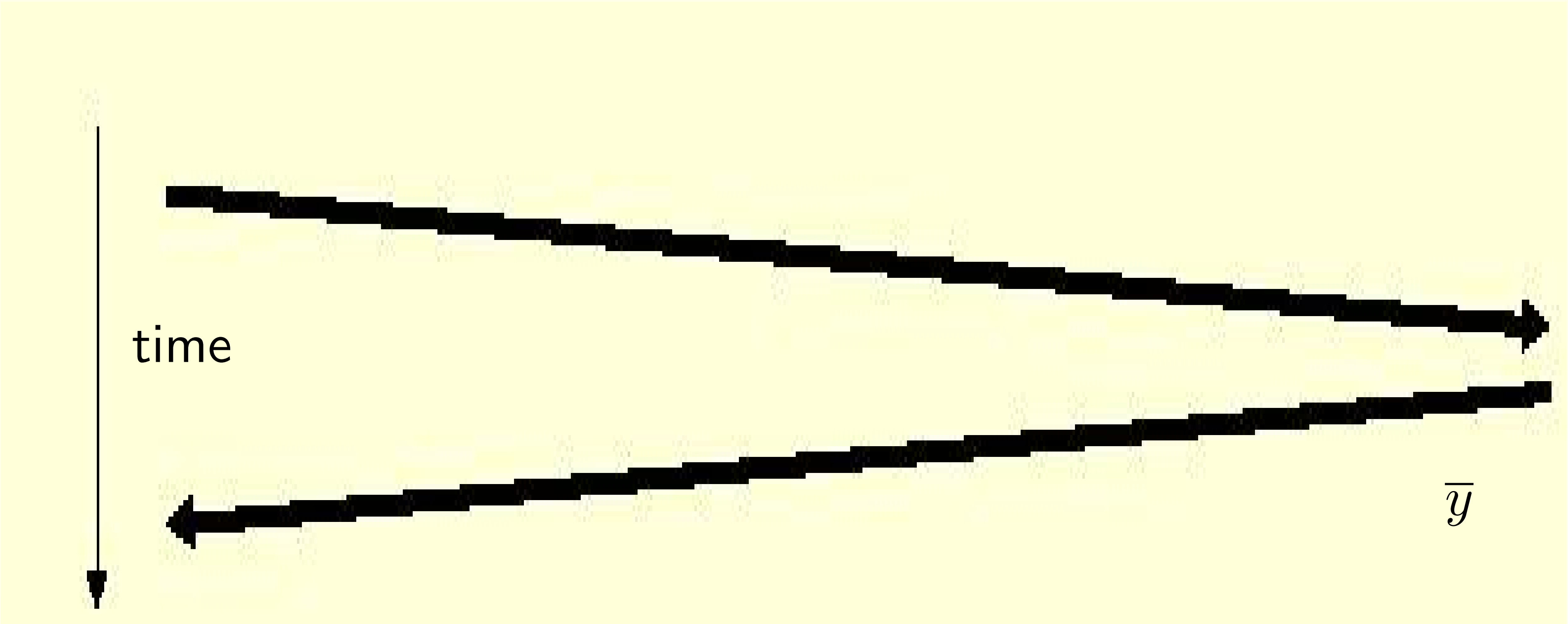
$$\dot{y} = f'(x).\dot{x} = f'_p(x_{p-1}).f'_{p-1}(x_{p-2}).\ldots.f'_1(x_0).\dot{x}$$

$$\begin{array}{lcl} & \ldots & \\ \dot{\mathbf{v}}_\mathbf{2} & = & \mathbf{2} * \dot{\mathbf{v}}_\mathbf{1} \\ \mathbf{v_2} & = & \mathbf{2} * \mathbf{v_1} + \mathbf{5} \\ \dot{\mathbf{v}}_\mathbf{4} & = & \dot{\mathbf{v}}_\mathbf{2} * (\mathbf{1} - \mathbf{p_1} * \mathbf{v_3}/\mathbf{v_2^2}) + \dot{\mathbf{v}}_\mathbf{3} * \mathbf{p_1}/\mathbf{v_2} \\ \mathbf{v_4} & = & \mathbf{v_2} + \mathbf{p_1} * \mathbf{v_3}/\mathbf{v_2} \\ & \ldots & \end{array}$$

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

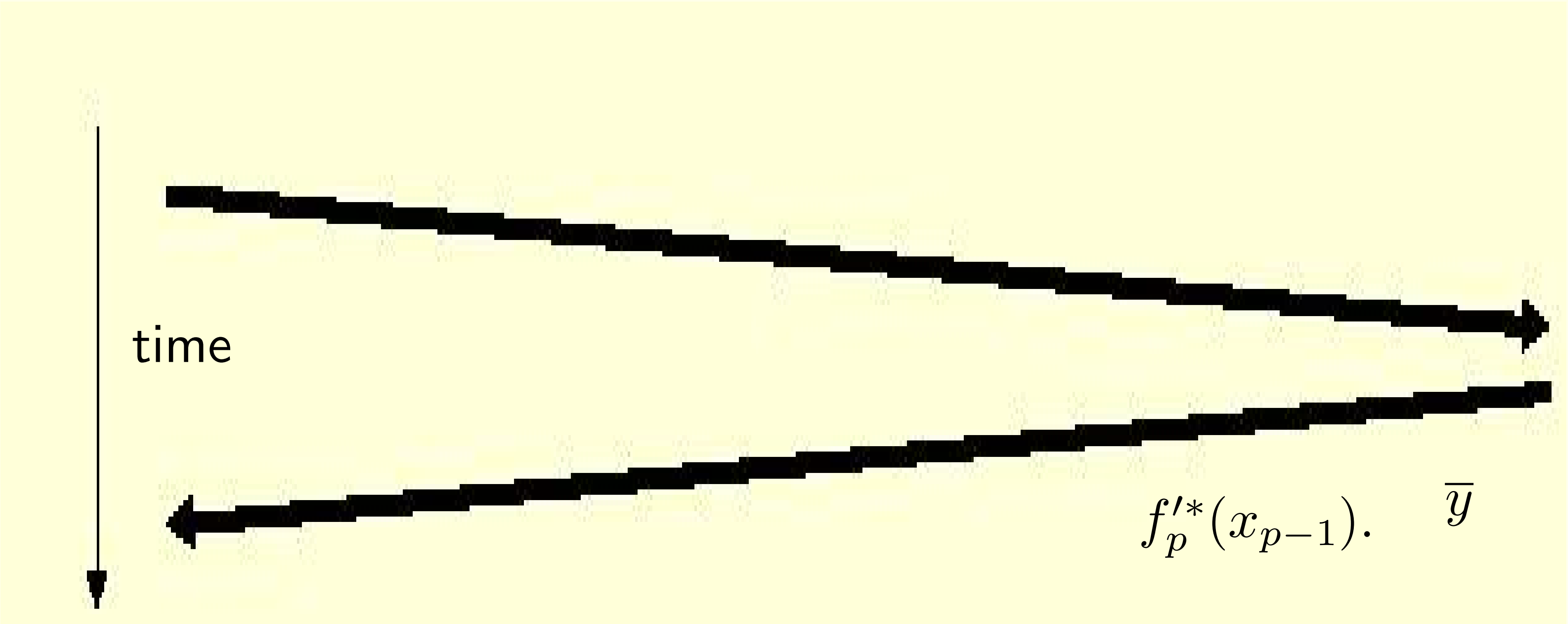
AD: Reverse is more tricky than Tangent

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



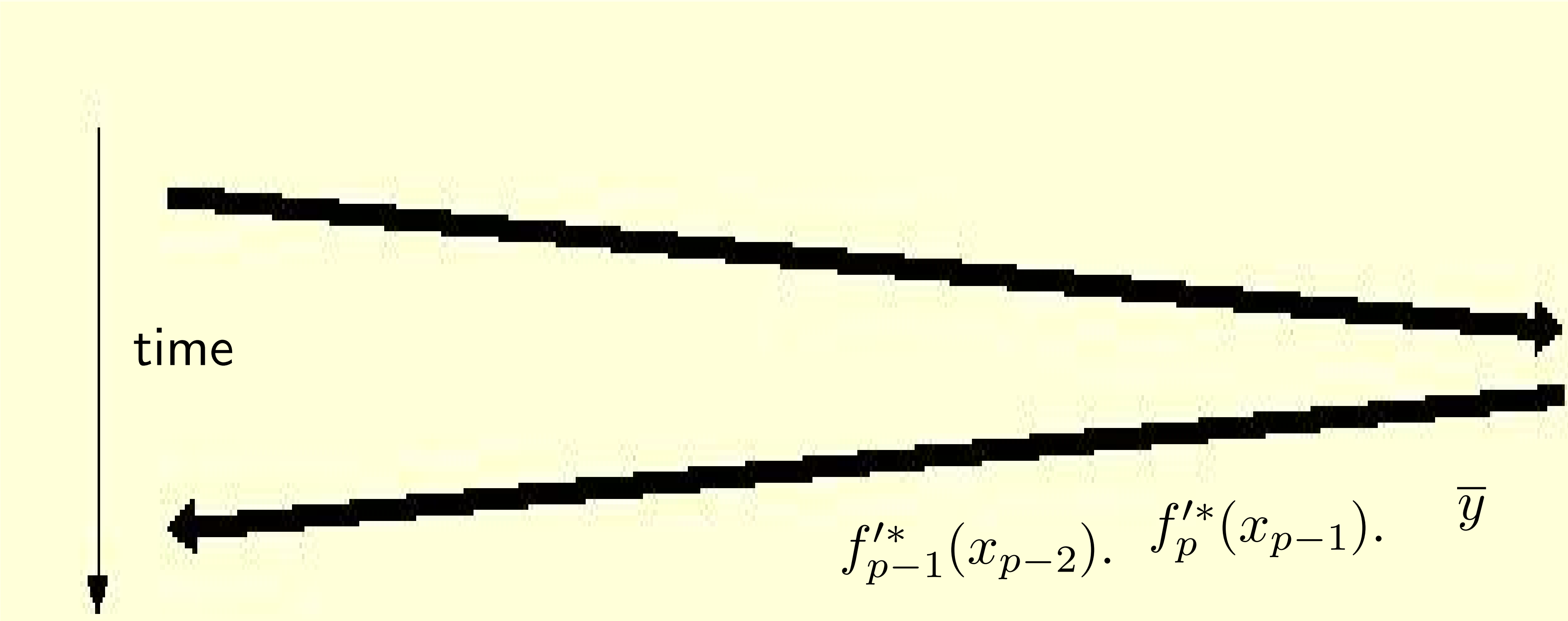
AD: Reverse is more tricky than Tangent

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



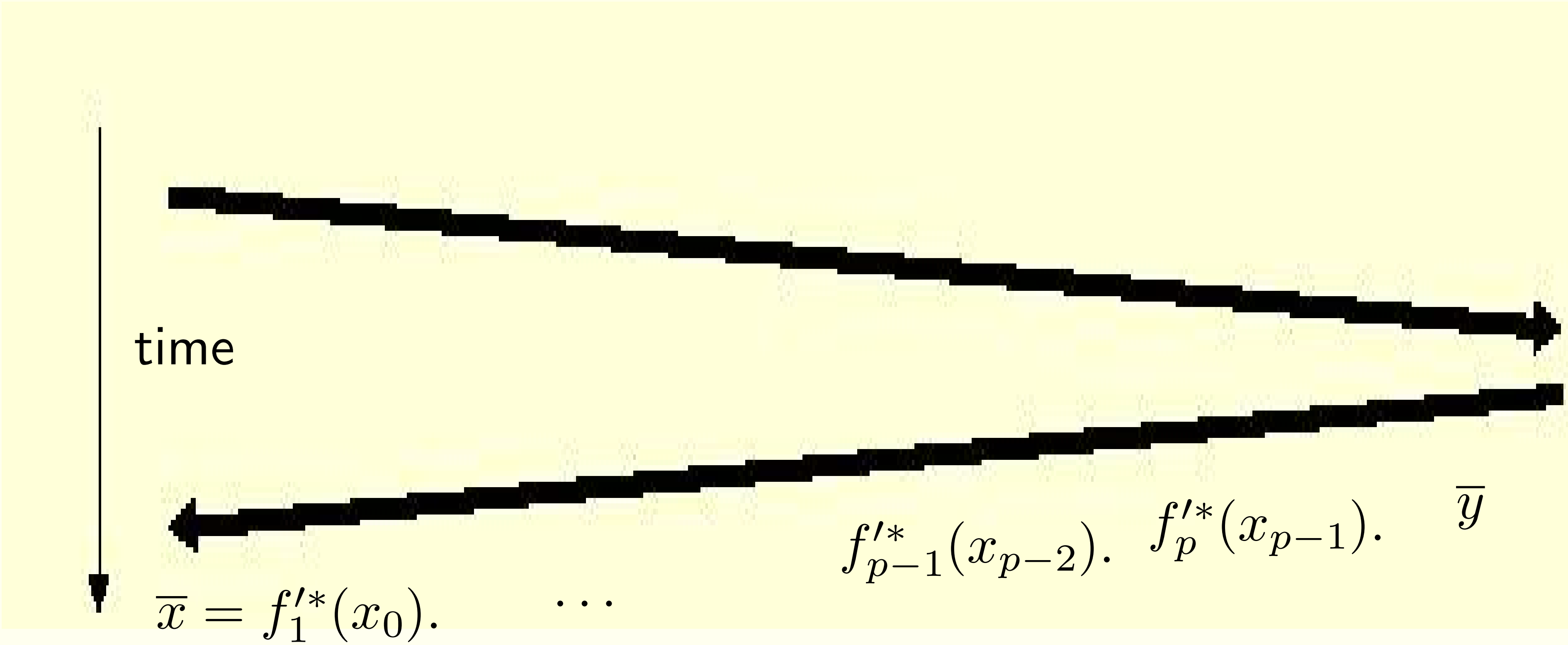
AD: Reverse is more tricky than Tangent

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



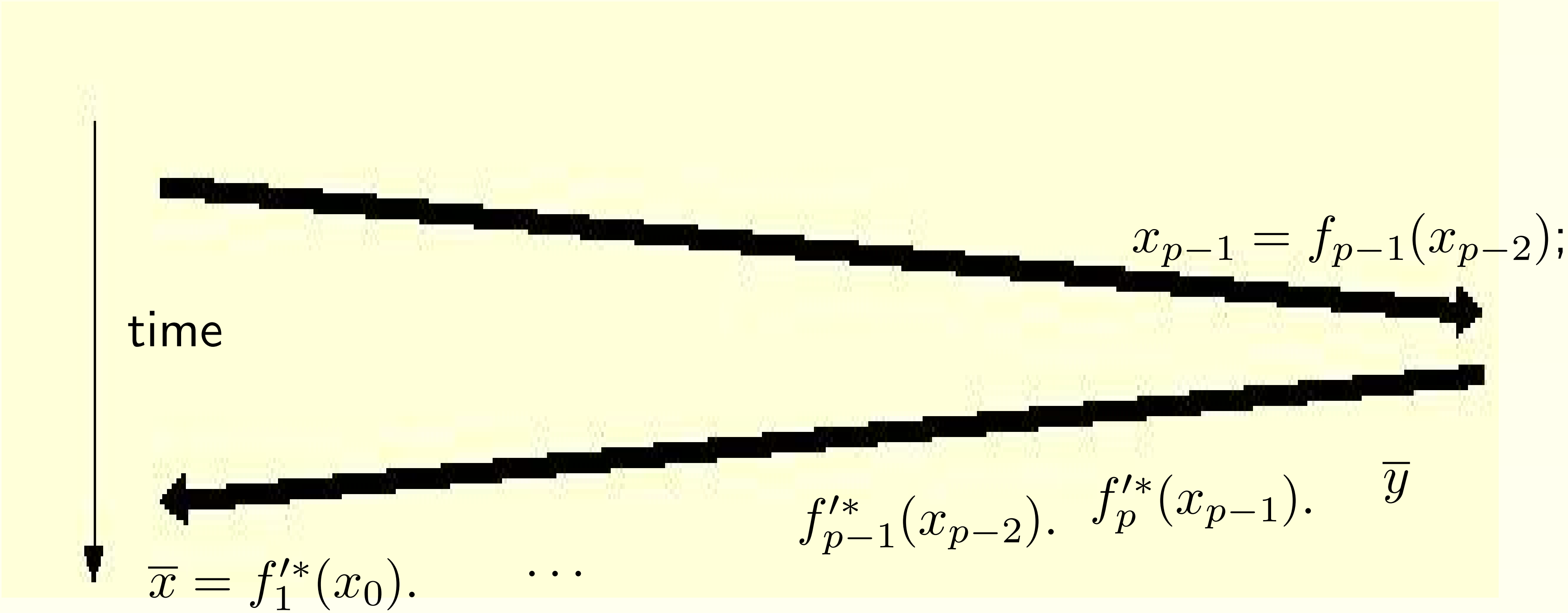
AD: Reverse is more tricky than Tangent

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



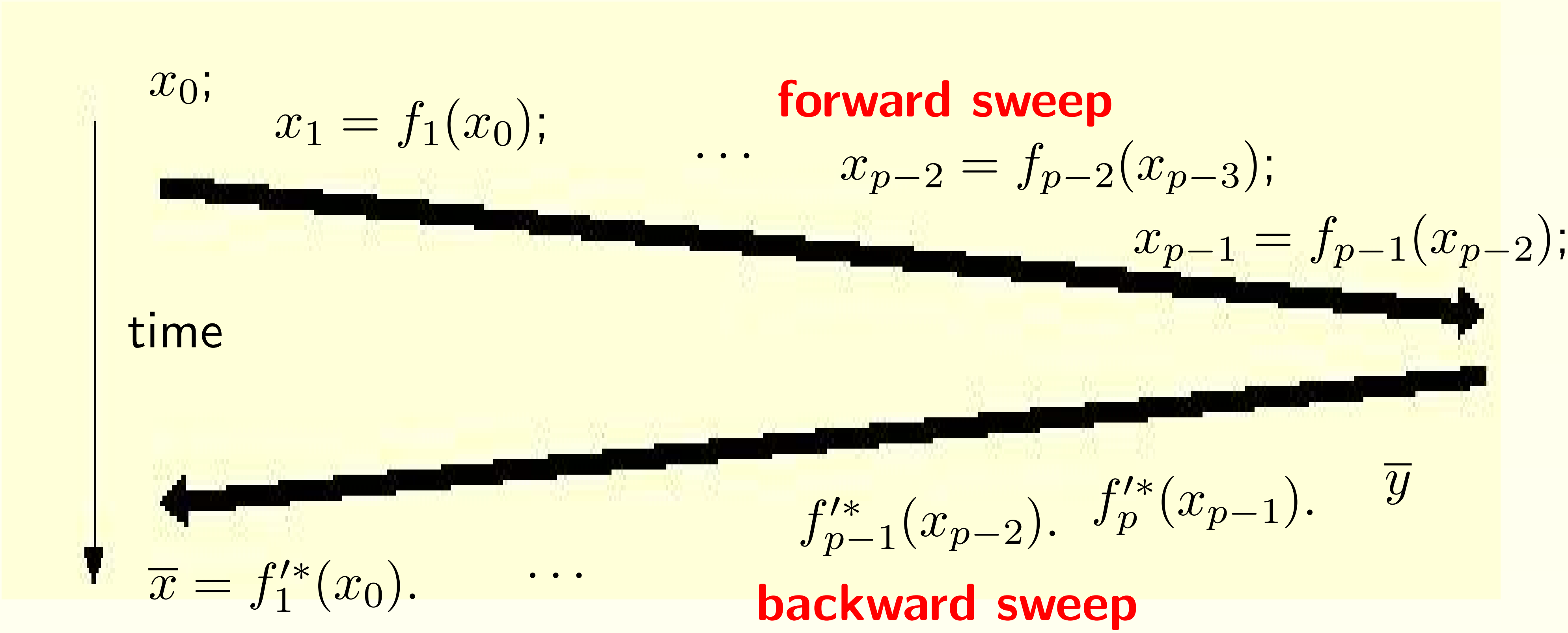
AD: Reverse is more tricky than Tangent

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



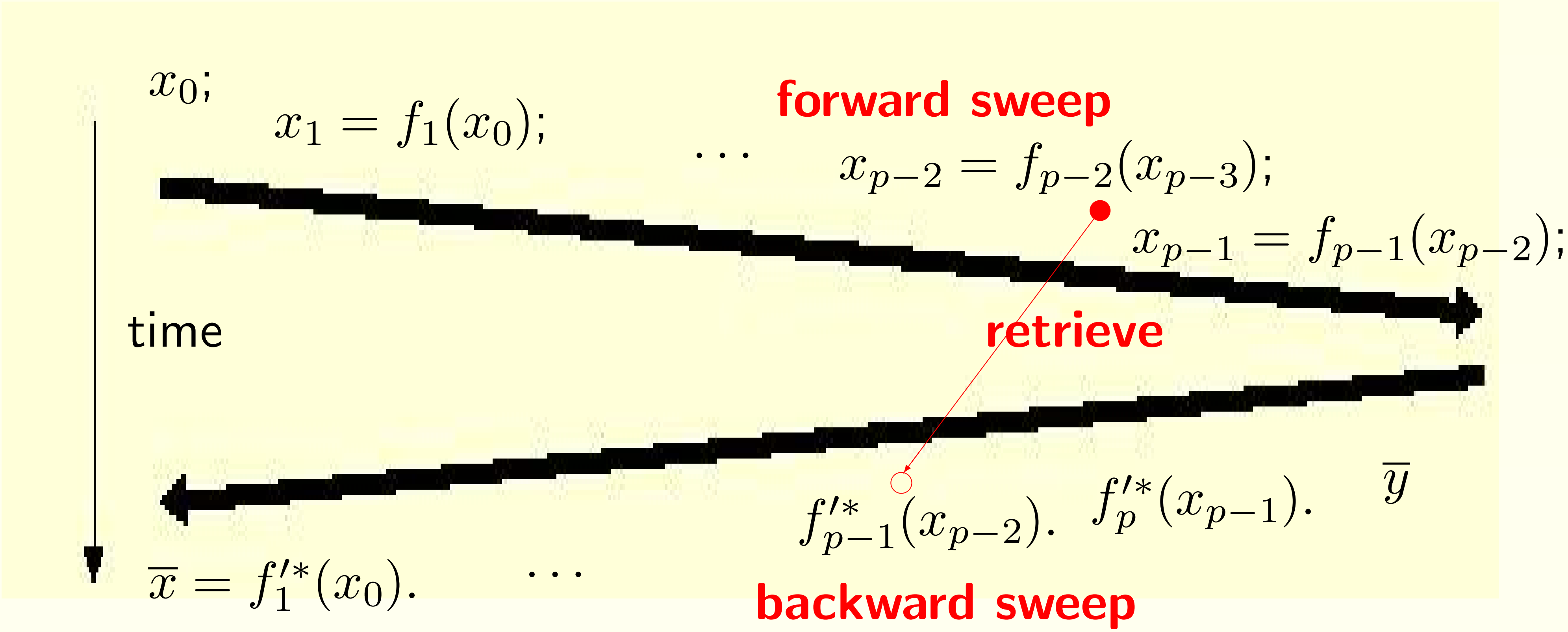
AD: Reverse is more tricky than Tangent

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



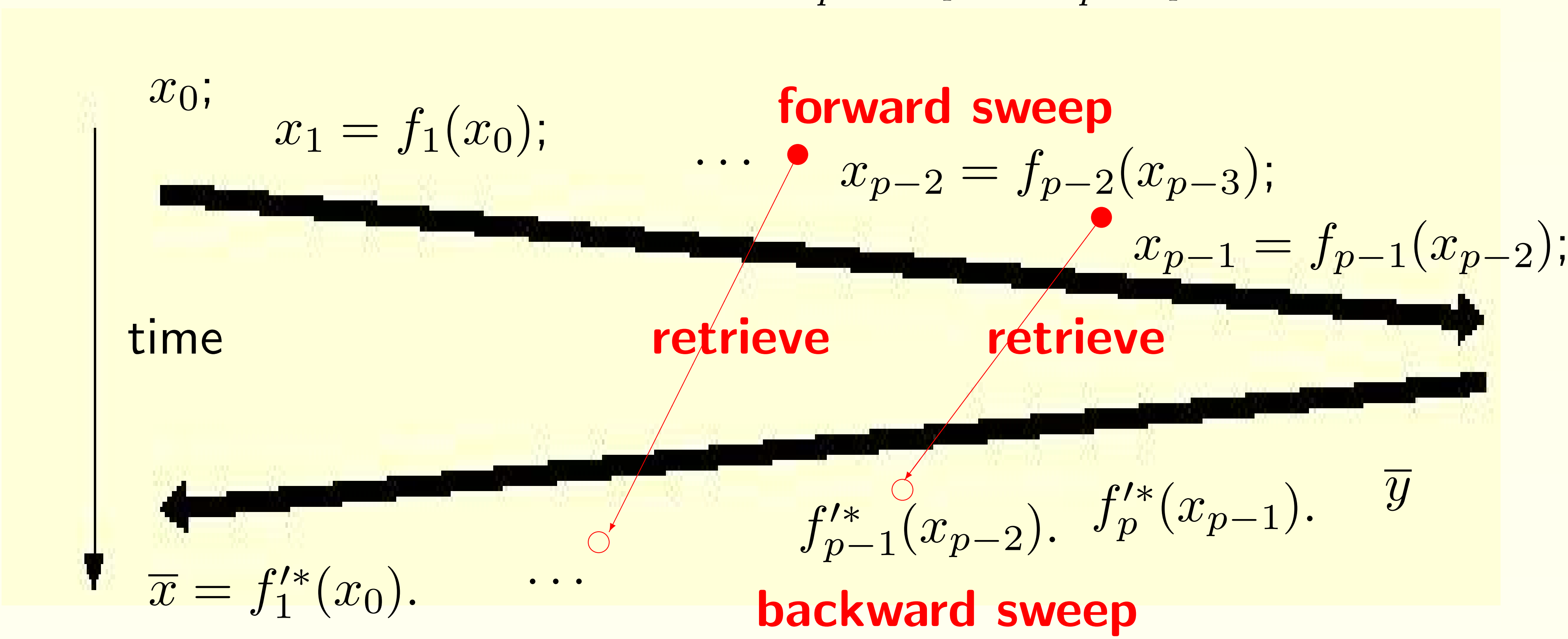
AD: Reverse is more tricky than Tangent

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



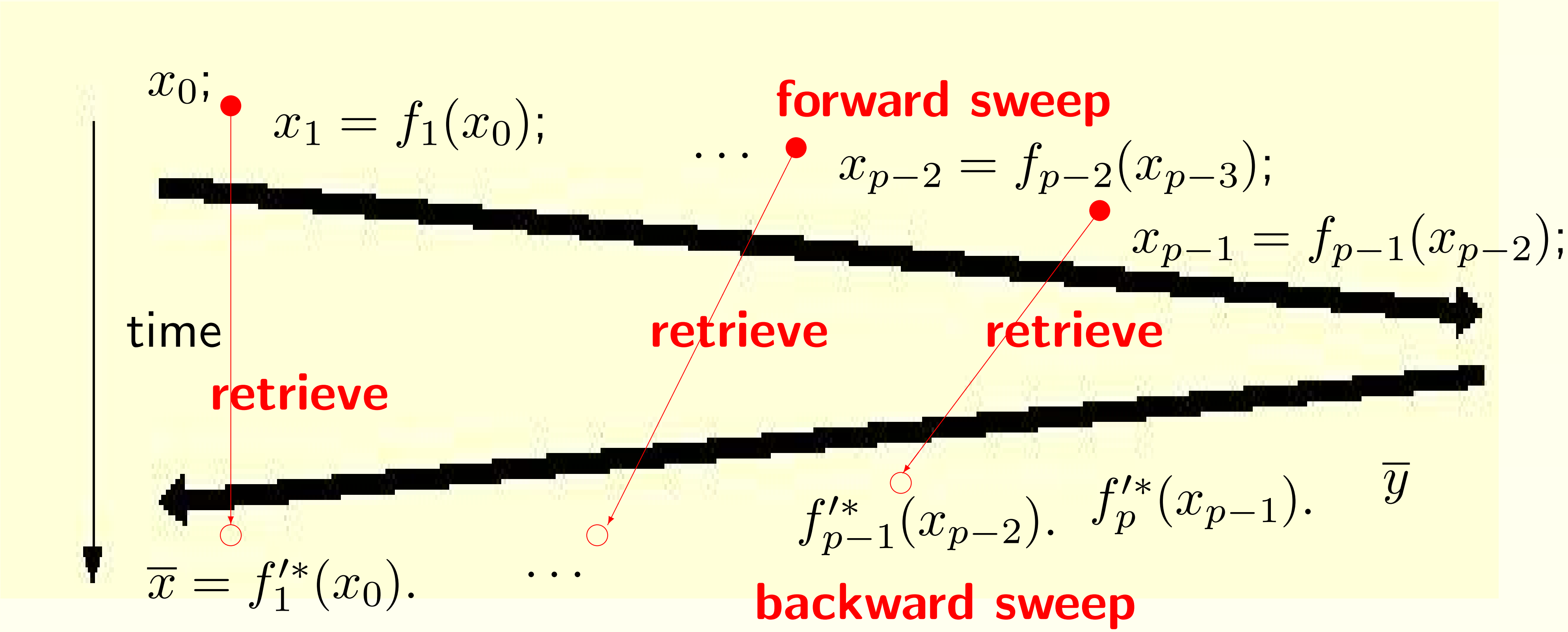
AD: Reverse is more tricky than Tangent

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



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:

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

AD: Continued Example

Program fragment:

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

Corresponding transposed Partial Jacobians:

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

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

$$\ldots$$

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

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

$$\ldots$$

$$\ldots$$

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

$$\bar{\mathbf{v}}_1 = \bar{\mathbf{v}}_1 + \mathbf{2} * \bar{\mathbf{v}}_2$$
$$\bar{\mathbf{v}}_2 = \mathbf{0}$$

$$\ldots$$

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)

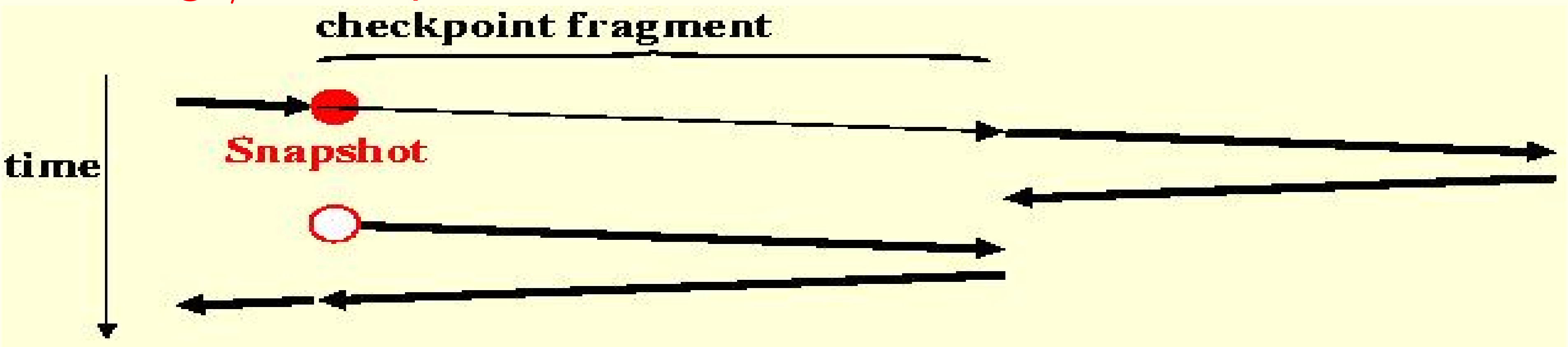
$$\bar{\mathbf{v}}_\mathbf{1} = \bar{\mathbf{v}}_\mathbf{1} + \mathbf{2} * \bar{\mathbf{v}}_\mathbf{2}$$

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

$$\ldots$$

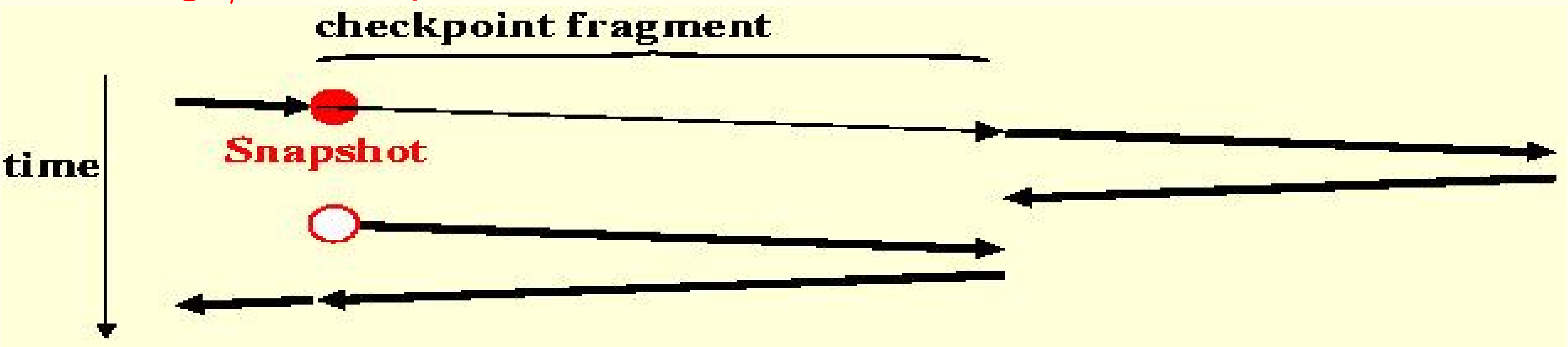
AD: The Checkpointing tactic

A Storage/Recomputation tradeoff:

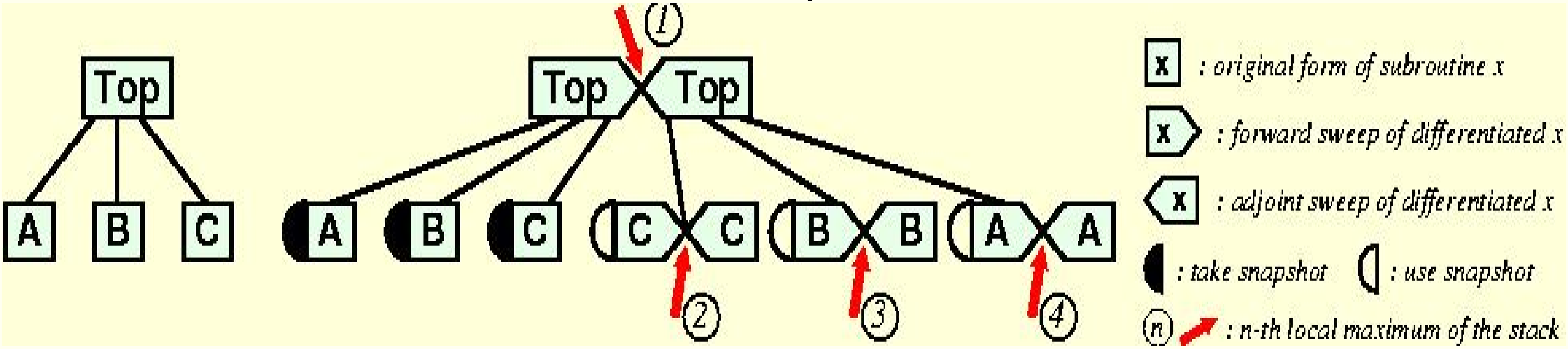


AD: The Checkpointing tactic

A Storage/Recomputation tradeoff:



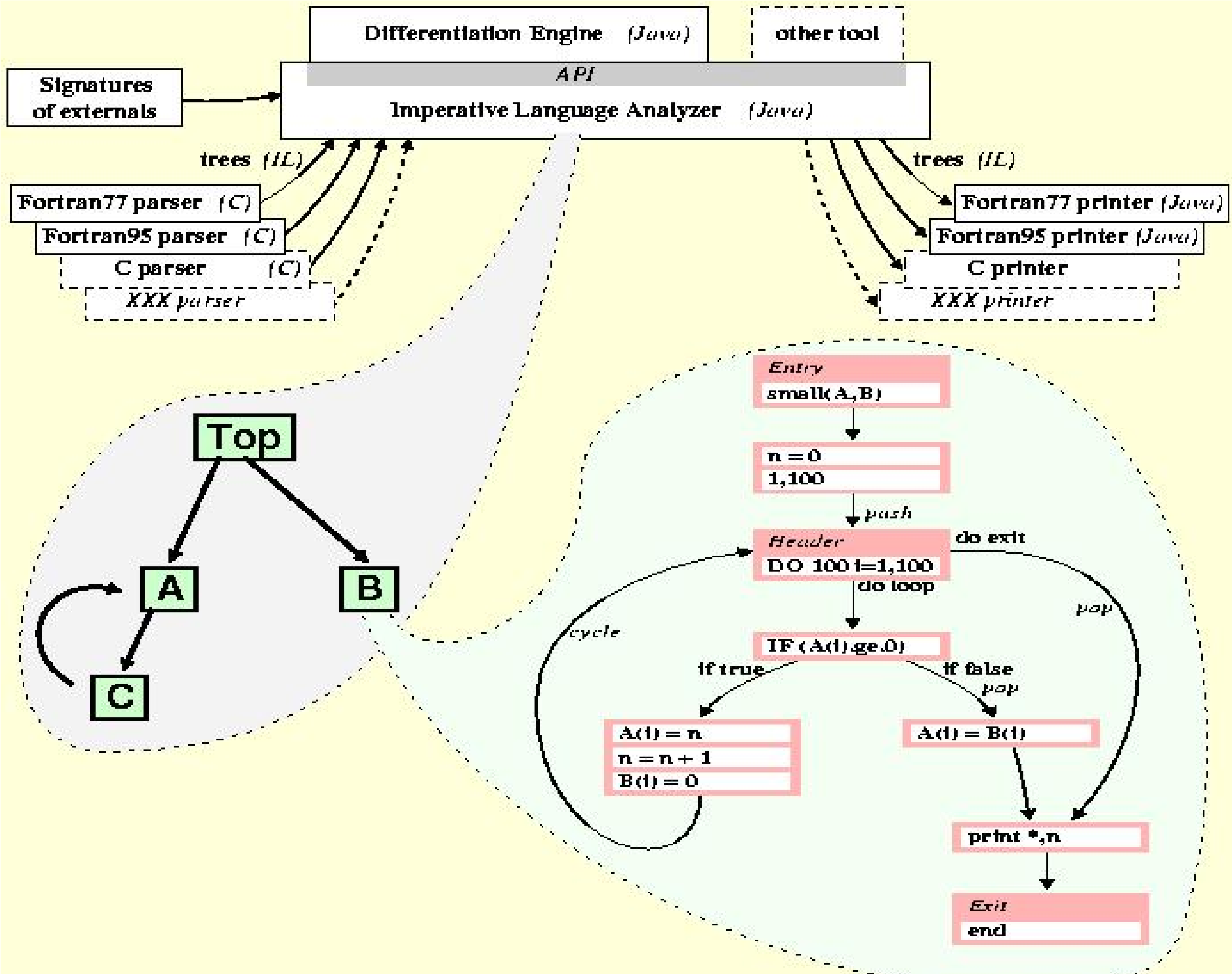
TAPENADE does it on the Call Graph :



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

Tapenade Modes

- normalize
- tangent
- multi-directional tangent
- reverse

Tapenade: Global Static Analyses on Flow Graphs

- forward (*resp.* backward) dependence wrt *independent inputs* (*resp. dependent inputs*)

- classical *IN-OUT* analysis (e.g. for snapshots)

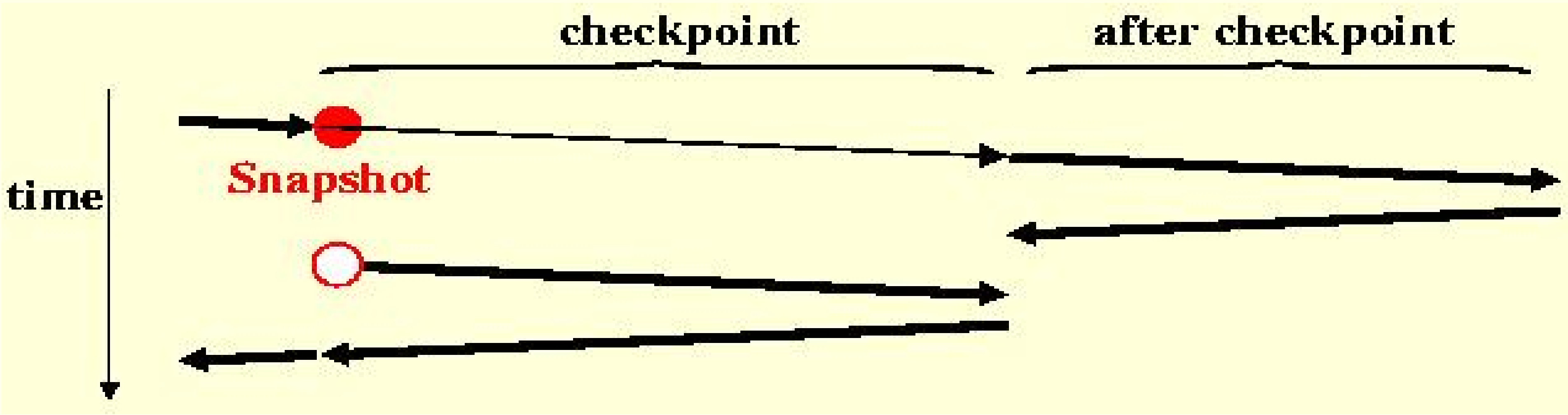
- specific for the reverse mode: *"To Be Restored"*, *Adjoint Dead Code*

- data-dependency analysis to reorder instructions.

- . . . *pointer* analysis . . .

Usual restrictions: conservative assumptions, arrays . . .

Example: reduced snapshots



$\textbf{Snapshot} = \textbf{IN}(\overline{\textbf{checkpoint}}) \bigcap \textbf{OUT}(\textbf{checkpoint and } \overline{\textbf{after}})$

Tapenade Differentiation model on examples

- Control structures
- Procedure calls and checkpointing
- "To Be Restored" analysis
- Instructions Reordering
- Dead Adjoint Code

Control structures

original program

```
SUBROUTINE S1(a, n, x)
...
DO i=2,n,7
 IF (a(i).GT.1.0) THEN
  a(i) = LOG(a(i)) + a(i-1)
  IF (a(i).LT.0.0) a(i)=2*a(i)
 END IF
ENDDO
```

Tapenade reverse: fwd sweep

```
DO i=2,n,7
 IF (a(i).GT.1.0) THEN
  CALL PUSHREAL4(a(i))
  a(i) = LOG(a(i))+a(i-1)
  IF (a(i).LT.0.0) THEN
   CALL PUSHREAL4(a(i))
   a(i) = 2*a(i)
   CALL PUSHINTEGER4(3)
  ELSE ...
```

Tapenade tangent

```
SUBROUTINE S1_D(a, ad, n, x)
...
DO i=2,n,7
 IF (a(i).GT.1.0) THEN
  ad(i)=ad(i)/a(i)+ad(i-1)
  a(i) = LOG(a(i)) + a(i-1)
  IF (a(i).LT.0.0) THEN
   ad(i) = 2*ad(i)
   a(i) = 2*a(i)
  END IF
```

Tapenade reverse: bwd sweep

```
CALL POPINTEGER4(adTo)
DO i=adTo,2,-7
 CALL POPINTEGER4(branch)
 IF (branch .GE. 2) THEN
  IF (branch .GE. 3) THEN
   CALL POPREAL4(a(i))
   ab(i) = 2*ab(i)
  END IF
  CALL POPREAL4(a(i))
  ab(i-1) = ab(i-1) + ab(i)
```

Procedure calls and checkpointing

<table>
<tr><th>original program</th><th>Tapenade reverse: fwd sweep</th></tr>
<tr><td><pre>x = x**3
CALL SUB(a, x, 1.5, z)
x = x*y</pre></td><td><pre>CALL PUSHREAL4(x)
x = x**3
CALL PUSHREAL4(x)
CALL SUB(a, x, 1.5, z)
x = x*y</pre></td></tr>
<tr><th>Tapenade tangent</th><th>Tapenade reverse: bwd sweep</th></tr>
<tr><td><pre>xd = 3*x**2*xd
x = x**3
CALL SUB_D(a, ad, x, xd,
           1.5, 0.0, z)
xd = y*xd
x = x*y</pre></td><td><pre>xb = y*xb
CALL POPREAL4(x)
CALL SUB_B(a, ab, x, xb,
           1.5, arg2b, z)
CALL POPREAL4(x)
xb = 3*x**2*xb</pre></td></tr>
</table>

"To Be Restored" analysis

original program	reverse mode: naive bwd sweep	reverse mode: bwd sweep with TBR
x = x + EXP(a) y = x + a**2 a = 3*z	CALL POPREAL4(a) zb = zb + 3*ab ab = 0.0 CALL POPREAL4(y) ab = ab + 2*a*yb xb = xb + yb yb = 0.0 CALL POPREAL4(x) ab = ab + EXP(a)*xb	CALL POPREAL4(a) zb = zb + 3*ab ab = 0.0 ab = ab + 2*a*yb xb = xb + yb yb = 0.0 ab = ab + EXP(a)*xb

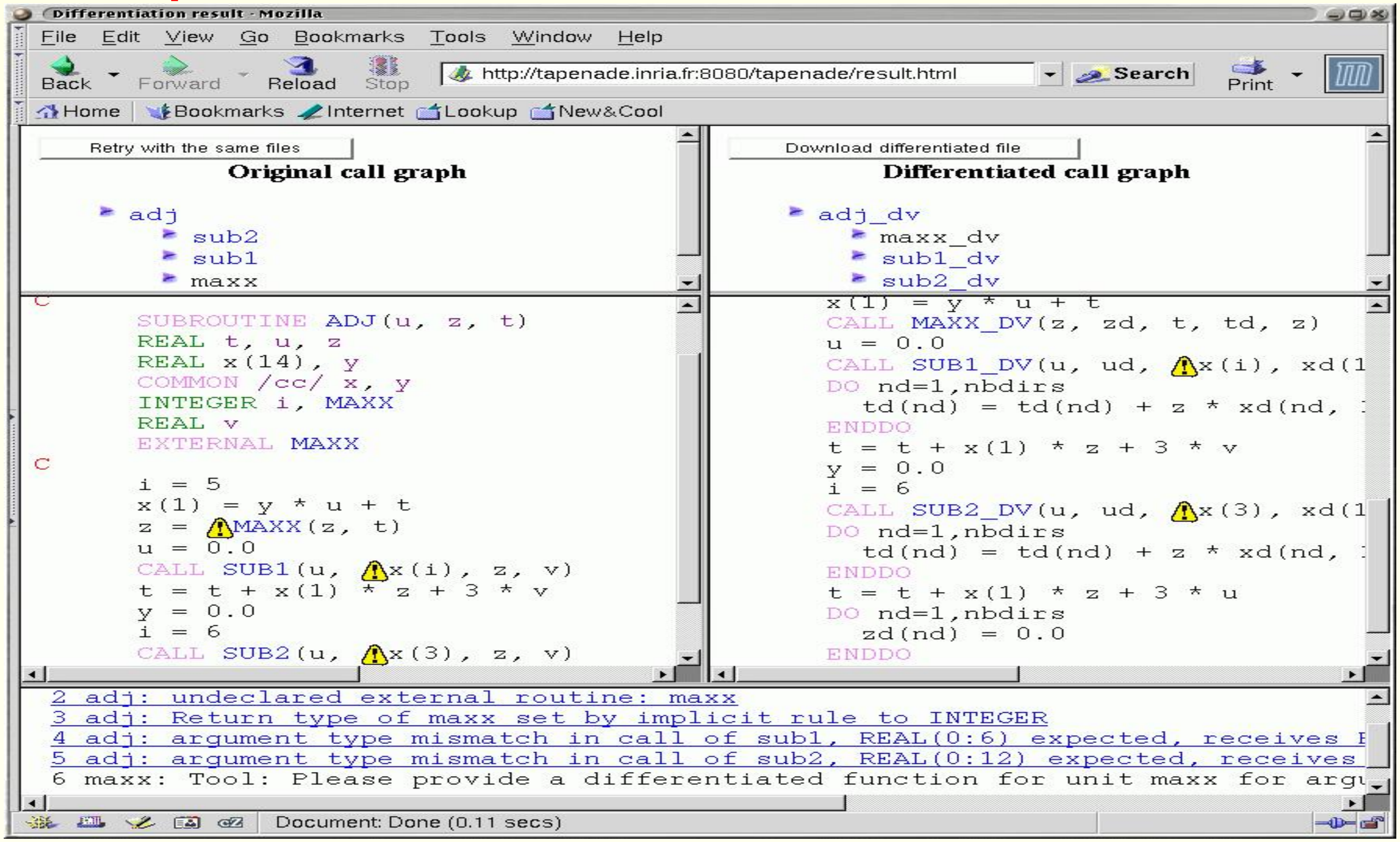
Instructions Reordering

original program	reverse mode: backward sweep with TBR	Tapenade reverse: non-incremental backward sweep
x = x + EXP(a) y = x + a**2 a = 3*z	CALL POPREAL4(a) zb = zb + 3*ab ab = 0.0 ab = ab + 2*a*yb xb = xb + yb yb = 0.0 ab = ab + EXP(a)*xb	CALL POPREAL4(a) zb = zb + 3*ab xb = xb + yb ab = 2*a*yb+EXP(a)*xb yb = 0.0

Dead Adjoint Code

original program	**reverse mode:**	**Tapenade reverse: ajd. dead code removed**
`IF (a.GT.0.0) THEN`	`IF (a .GT. 0.0) THEN`	`IF (a .GT. 0.0) THEN`
`  a = LOG(a)`	`  CALL PUSHREAL4(a)`	
`ELSE`	`  a = LOG(a)`	
`  a = LOG(c)`	`  CALL POPREAL4(a)`	
`  CALL SUB(a)`	`  ab = ab/a`	`  ab = ab/a`
`ENDIF`	`ELSE`	`ELSE`
`END`	`  a = LOG(c)`	`  a = LOG(c)`
	`  CALL PUSHREAL4(a)`	
	`  CALL SUB(a)`	
	`  CALL POPREAL4(a)`	
	`  CALL SUB_B(a, ab)`	`  CALL SUB_B(a, ab)`
	`  cb = cb + ab/c`	`  cb = cb + ab/c`
	`  ab = 0.0`	`  ab = 0.0`
	`END IF`	`END IF`

Tapenade: an AD tool on the web



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

Conclusion and future work

Tapenade now 3 years old.

Several applications on industrial and academic codes:
Aeronautics, Hydrology, Chemistry, Biology, Agronomy...

Future developments:

- In progress: `FORTRAN95`, and then `C` ($\Rightarrow$ pointers!)
- User Directives: active I-O, checkpoints, special loops
- Validity domain for derivatives