For-loop equivalents and massive parallelization in CasADi

Citius, Altius, Fortius

Joris Gillis
(MECO group, KU Leuven)

Outline

- Key ingredients of CasADi
- New concepts: Map, MapAccum
- Massive parallelization
- Software architecture

Core idea

- Computer-aided formulation of optimization problems

symbolic primitive

expression

```
x = optivar()
a = optivar()

optimize(x^2+a^2, [ x+a^2>= 0 ])
```

$$\underset{x,a}{\text{minimize}} \quad x^2 + a^2$$
$$\text{subject to} \quad x + a^2 \geq 0$$

Obtain gradients, Jacobians, Hessians automatically

Basic example

- Find initial condition to achieve a goal

$$\dot{x} = a\,x + \cos x$$

$$\underset{x_0}{\text{minimize}} \quad x_N^2$$

```
x0 = optivar()
a  = optivar()
dt = 1


x1 = x0 + dt*(a*x0 + cos(x0))
x2 = x1 + dt*(a*x1 + cos(x1))
x3 = x2 + dt*(a*x2 + cos(x2))
…
xN = …

optimize(xN**2)
```

Basic example

- Find initial condition to achieve a goal

$$\dot{x} = a\,x + \cos x$$

$$\underset{x_0}{\text{minimize}} \quad x_N^2$$

```
x0 = optivar()
a  = optivar()
dt = 1


x = x0
for i in range(N):
    x = x + dt*(a*x + cos(x))

xN = x

optimize(xN**2)
```

Key ingredient: efficient symbolics

Matlab symbolic toolbox (mupad)

```
syms('x0')
syms('a')
dt = 1;

x = x0;

for i=1:N
  x = x+dt*(a*x+cos(x));
end
```

Key ingredient: efficient symbolics

Matlab: yalmip

```
x0 = sdpvar(1,1)
a = sdpvar(1,1)
dt = 1;

x = x0;

for i=1:N
  x = x+dt*(a*x+cos(x));
end
```

Key ingredient: efficient symbolics

Python: sympy

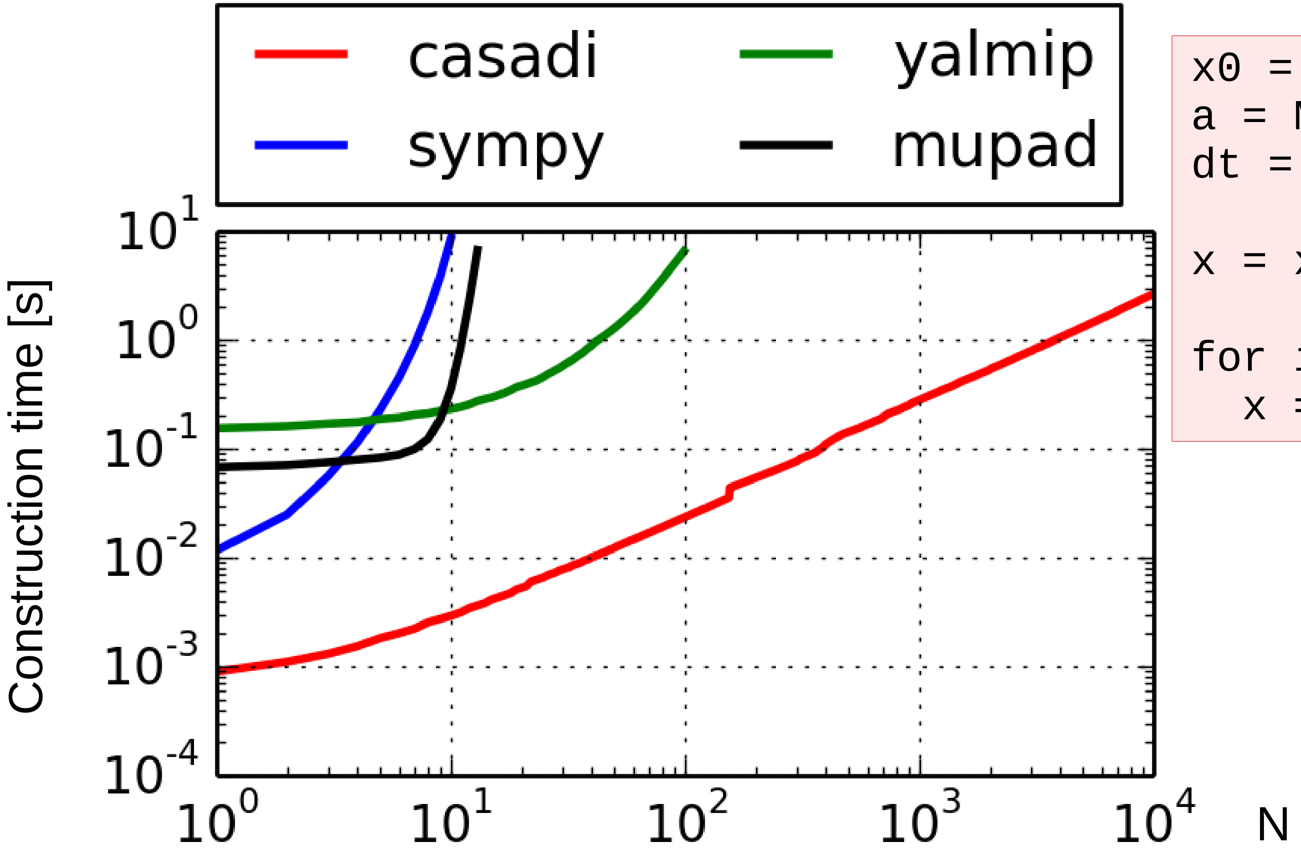
```
x0 = symbols('x0')
a = symbols('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))
```

Key ingredient: efficient symbolics

Python: casadi

```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))
```

Key ingredient: efficient symbolics

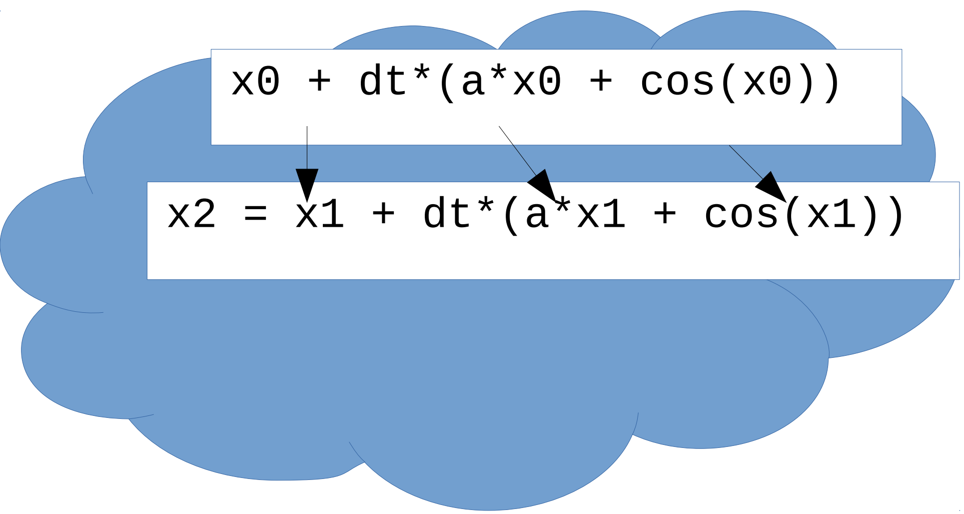



```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))
```

KU LEUVEN

Key ingredient: efficient symbolics

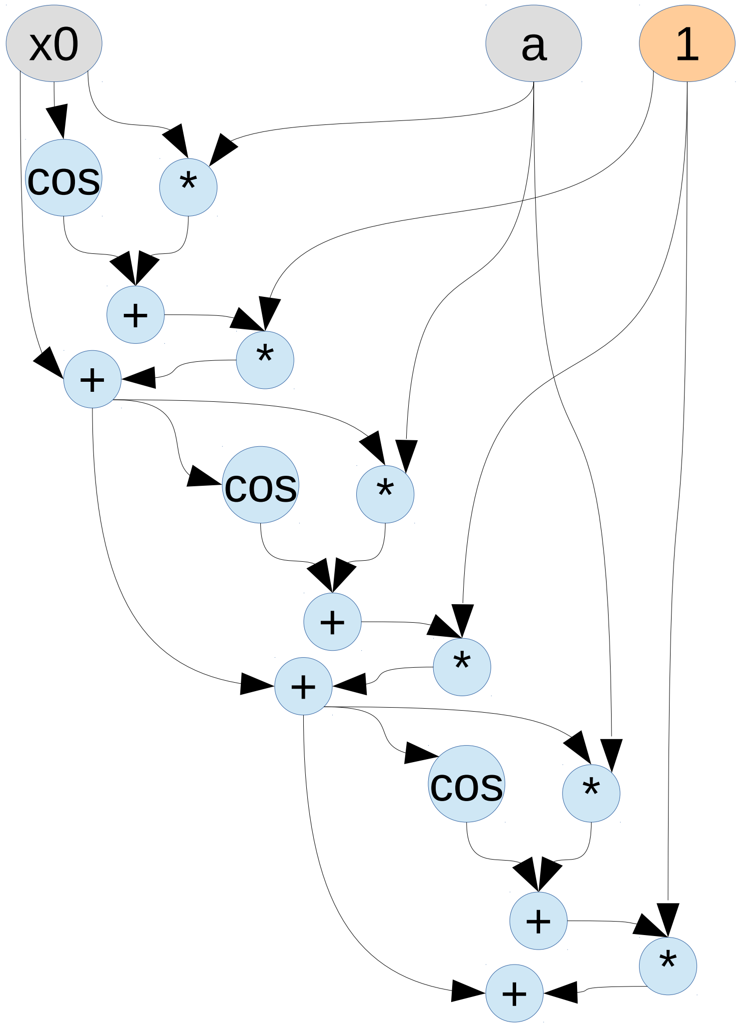
```
x0 = optivar()
a    = optivar()
dt   = 1


x1 = x0 + dt*(a*x0 + cos(x0))
x2 = x1 + dt*(a*x1 + cos(x1))
x3 = x2 + dt*(a*x2 + cos(x2))
…
xN = …
```




```
x2 = (x0 + dt*(a*x0 + cos(x0))) + dt*(a*(x0 + dt*(a*x0 + cos(x0))) + cos(x0 + dt*(a*x0 + cos(x0))))
```

Expression length ~ 3^N

Key ingredient: efficient symbolics



graph representation

```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))
```

#nodes: O(N)

KU LEUVEN

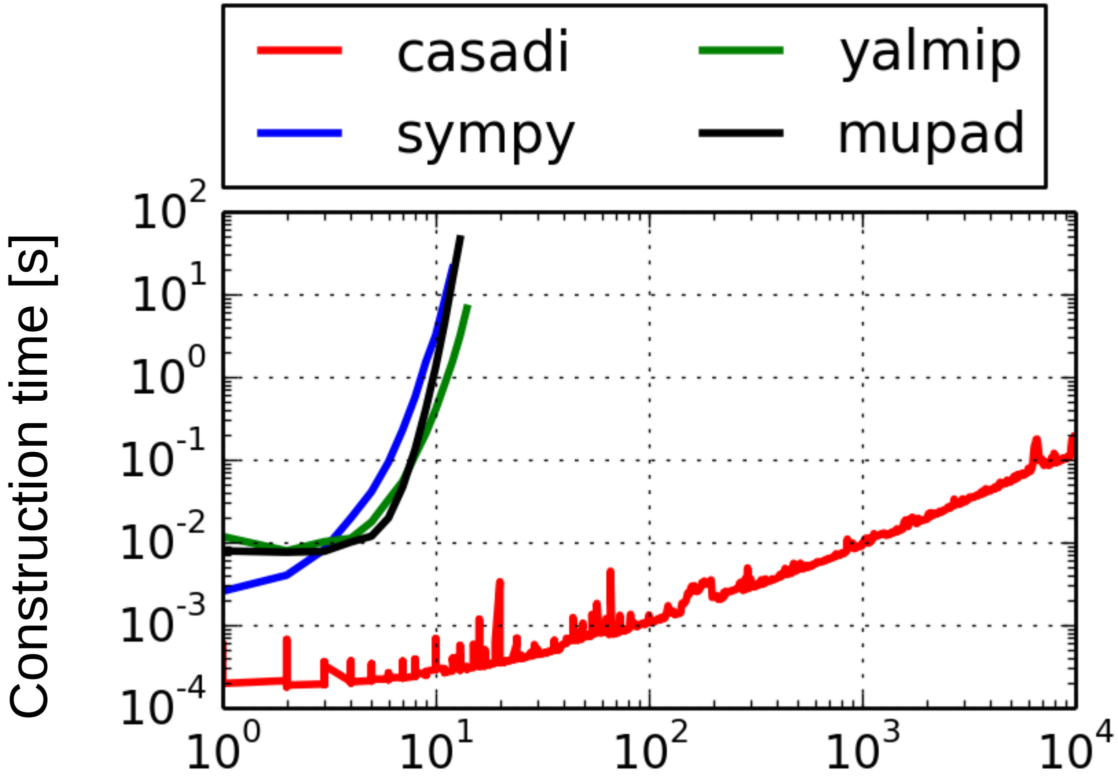
Key ingredient: algorithmic differentiation

Cost of a (forward/reverse) derivative of:
small multiple of original

$$\underset{x_0}{\text{minimize}} \quad x_N^2$$

$$\frac{d\,x_N}{d\,x_0}$$

Key ingredient: algorithmic differentiation




```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))

jacobian(x,x0)
```

$$\frac{d x_N}{d x_0}$$

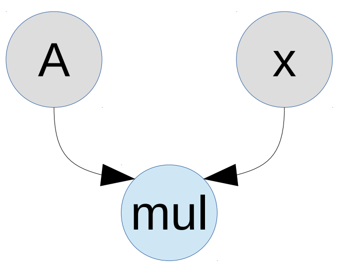
Key ingredient: matrix-valued graphs

$$\dot{x} = A\,x + \cos x$$

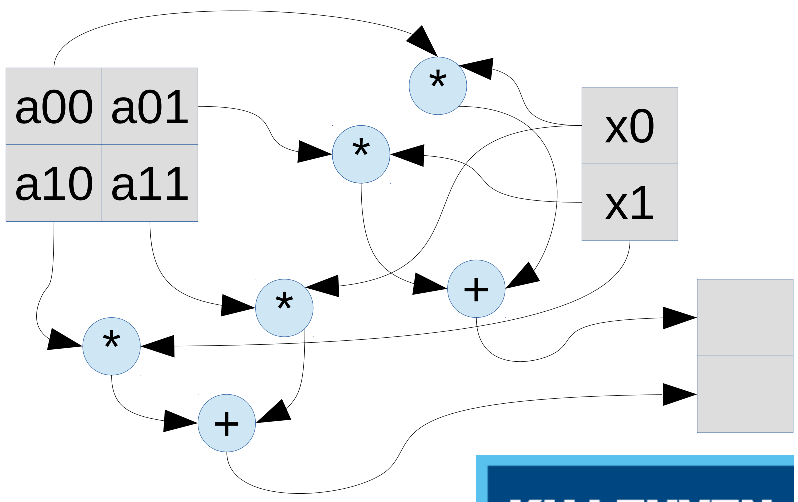
$$x \in \mathbb{R}^n$$

```
x = MX.sym('x',2,2)
A = MX.sym('A',2)

mul(A,x)
```

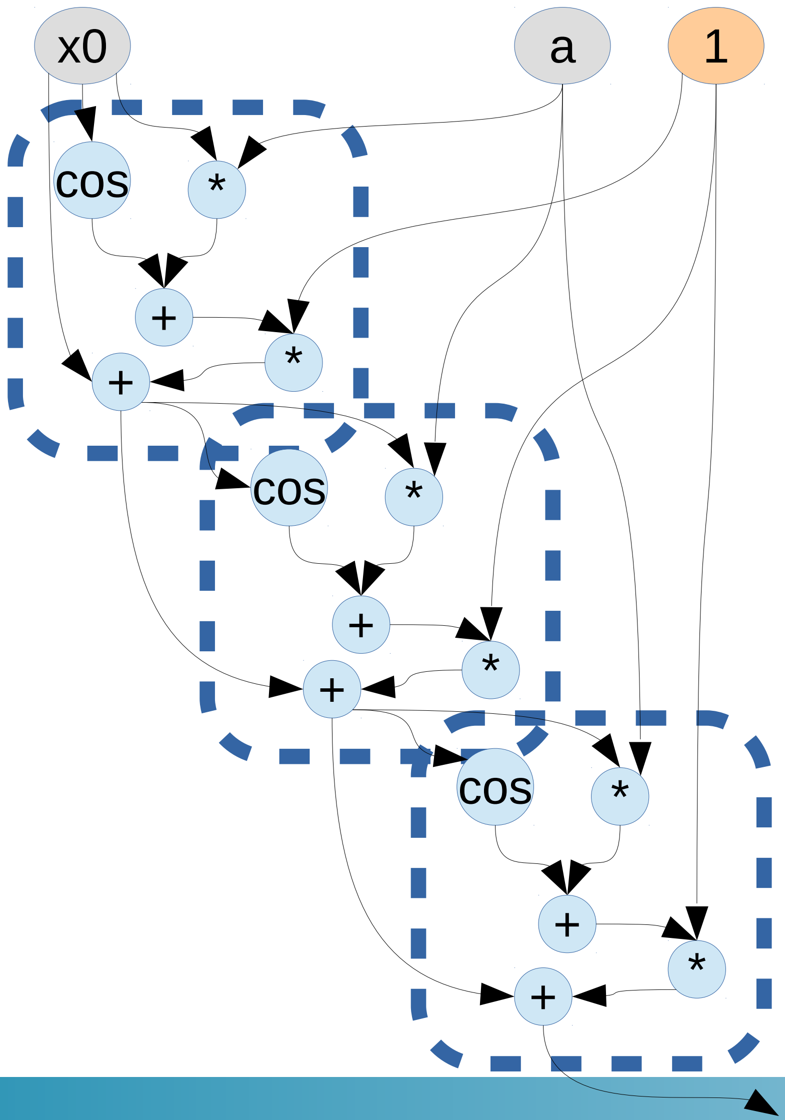
```
x = SX.sym('x',2,2)
A = SX.sym('A',2,2)

mul(A,x)
```



Scalar expansion
~ loop unrolling



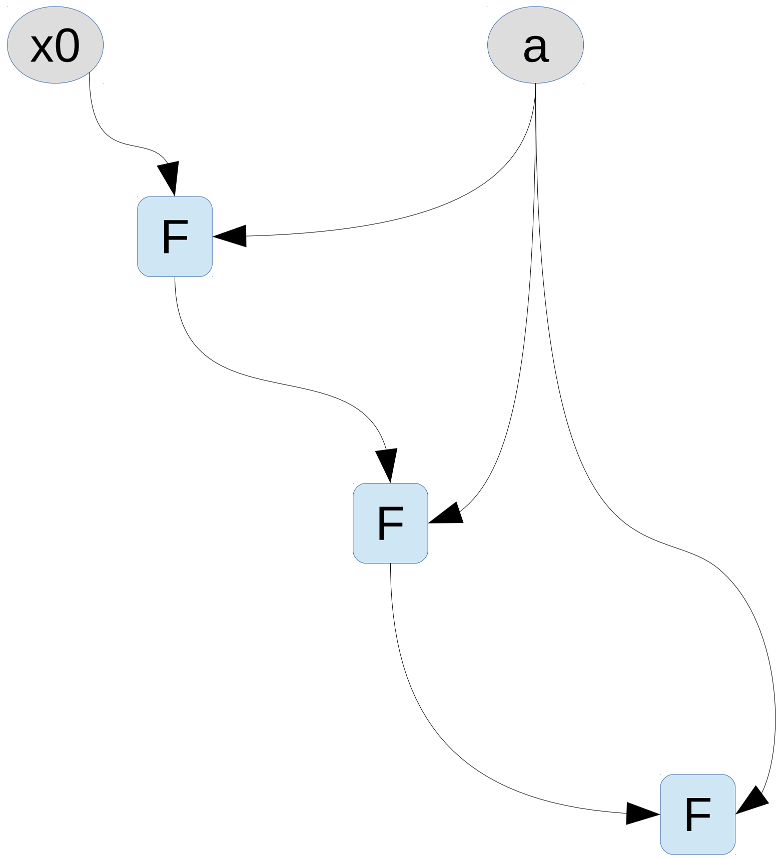
Key ingredient: function embedding




```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = x+dt*(a*x+cos(x))
```

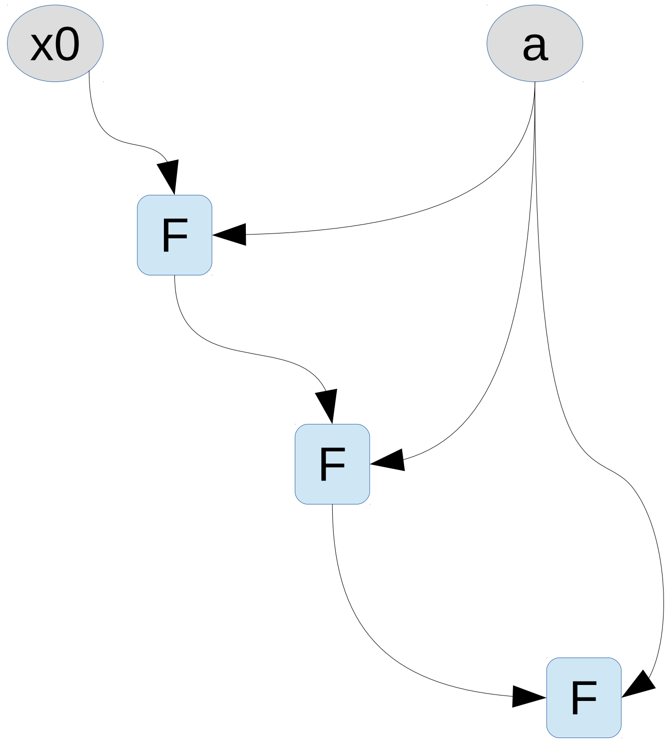
KU LEUVEN

Key ingredient: function embedding

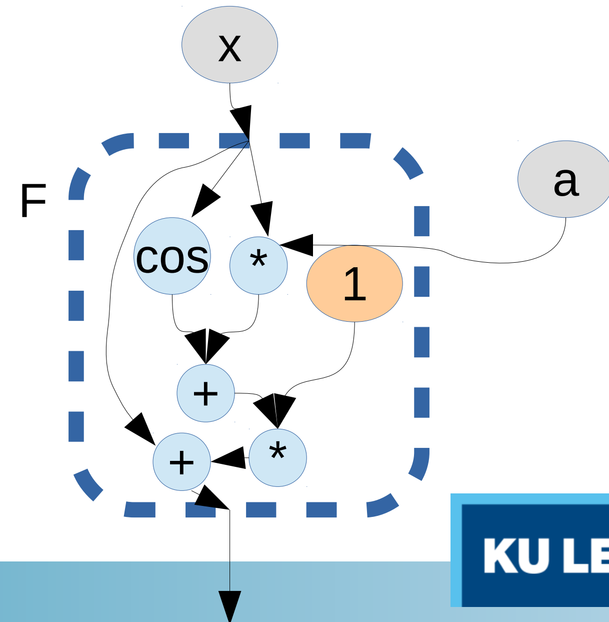



```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = F(x,a)
```

Key ingredient: function embedding




```
x0 = MX.sym('x0')
a = MX.sym('a')
dt = 1

x = x0

for i in range(N):
  x = F(x,a)
```




```
x = MX.sym('x')
a = MX.sym('a')
dt = 1

e = x+dt*(a*x+cos(x))
F = Function("F",[x,a],[e])
```

KU LEUVEN

Key ingredient: function embedding

rootfinder
linsol
integrator
...

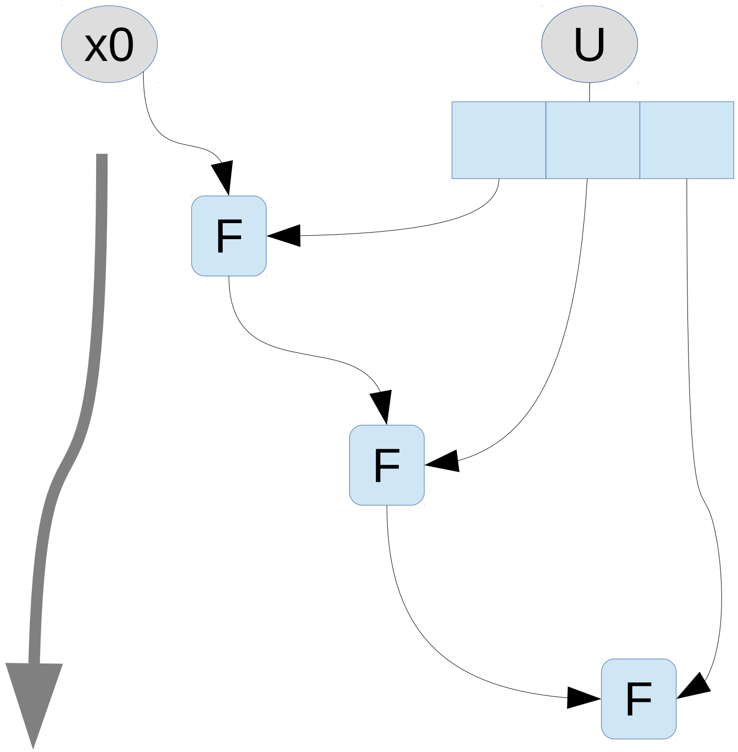
Outline

- Key ingredients of CasADi
  - Efficient symbolics
  - Algorithmic differentiation
  - Matrix-valued graphs
  - Function embedding
- New concepts: Map, MapAccum
- Massive parallelization
- Software architecture

KU LEUVEN

Towards large scale optimal control

$$x_{k+1} = F(x_k, u_k)$$

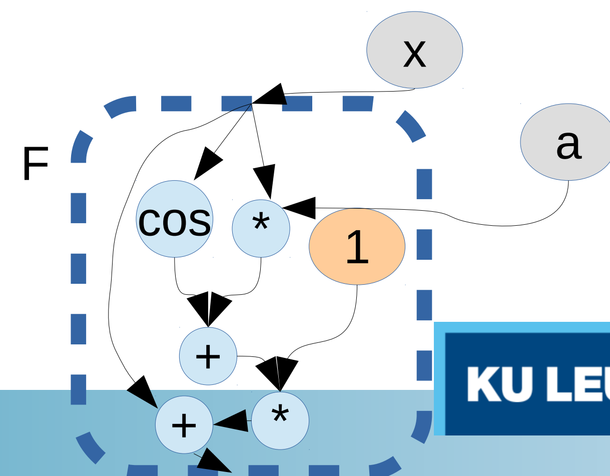


$$\underset{x_{0,} u_{0,} u_1 \ldots u_{N-1}}{\text{minimize}} \quad x_N^2$$

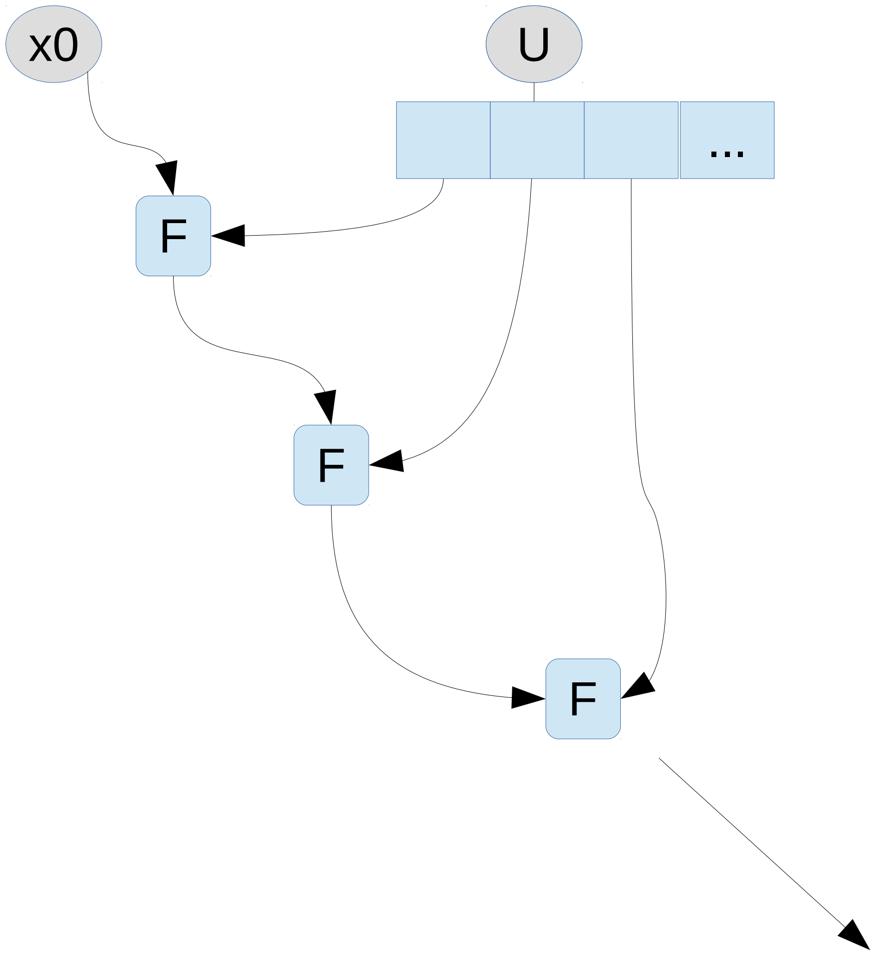
```
x0 = MX.sym('x0')
U = MX.sym('u',1,N)
dt = 1

w = horzsplit(U)

x = x0

for i in range(N):
  x = F(x,w[i])
```

MX graph (optimized for memory footprint)

SX graph (optimized for speed)



Towards large scale optimal control

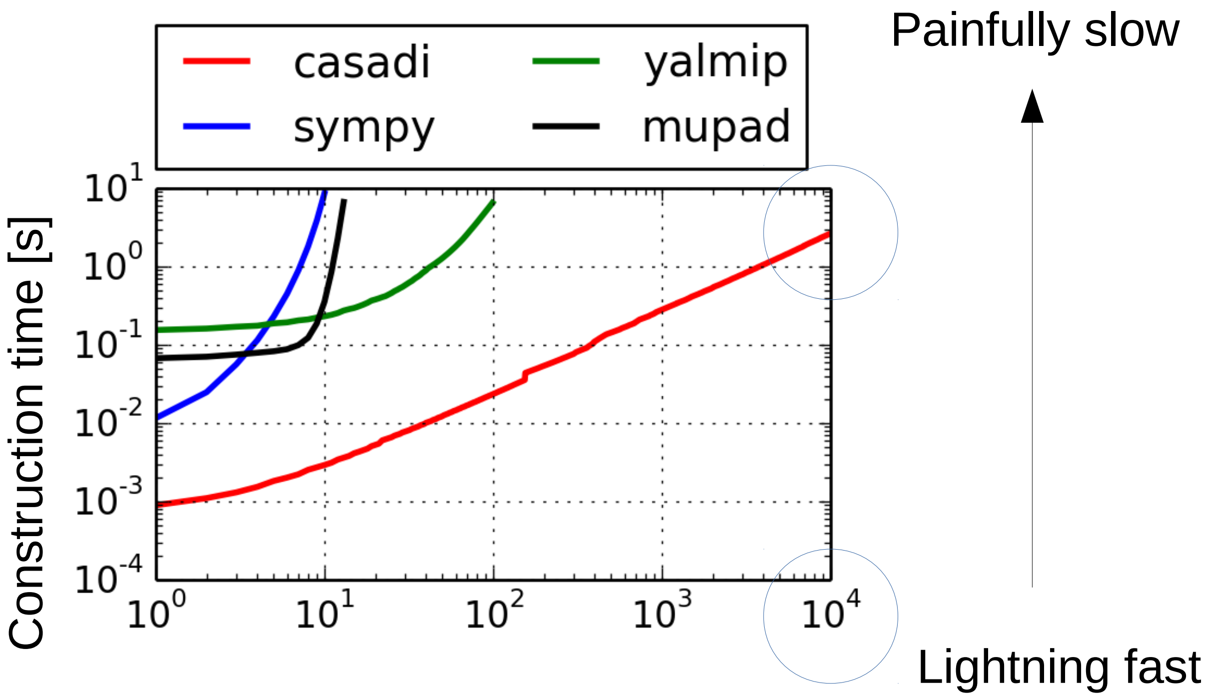


N = 10 000 , 100 000, ...

KU LEUVEN

Towards large scale optimal control

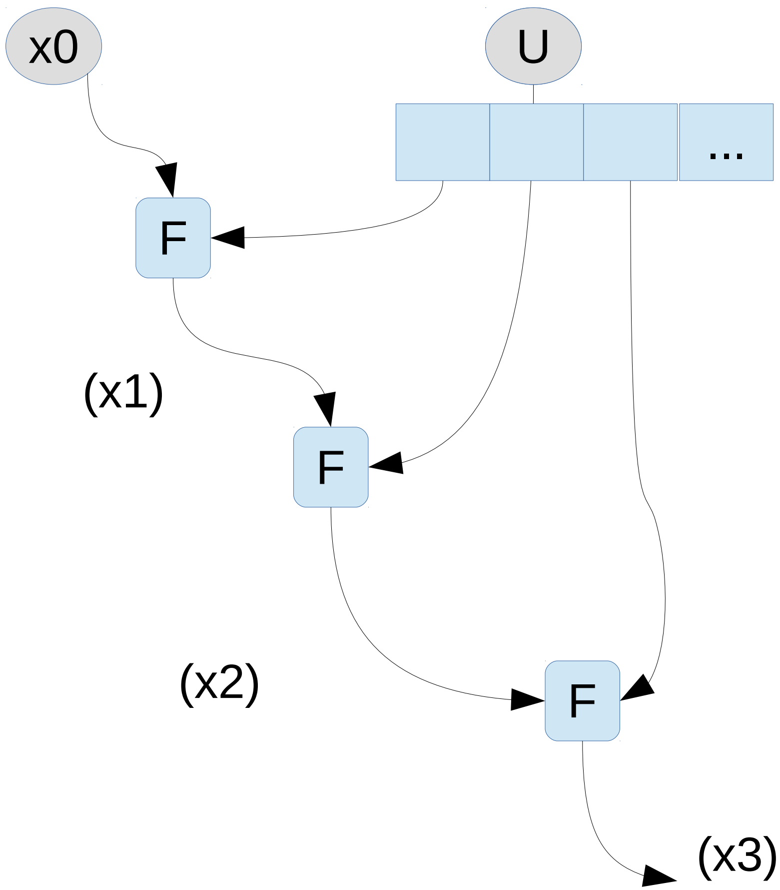
Not fast enough!

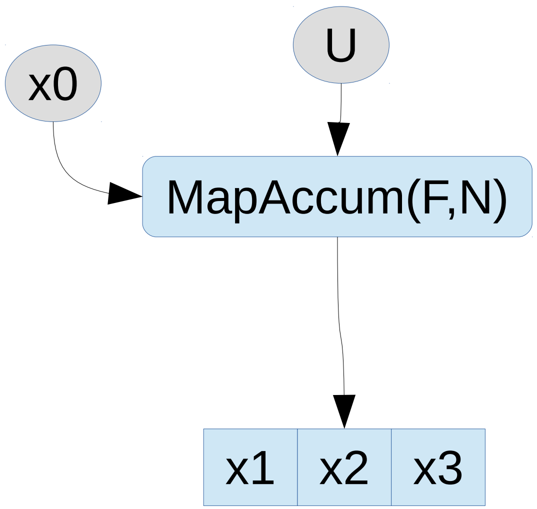


N = 10 000 , 100 000, ...

KU LEUVEN

New node: MapAccum



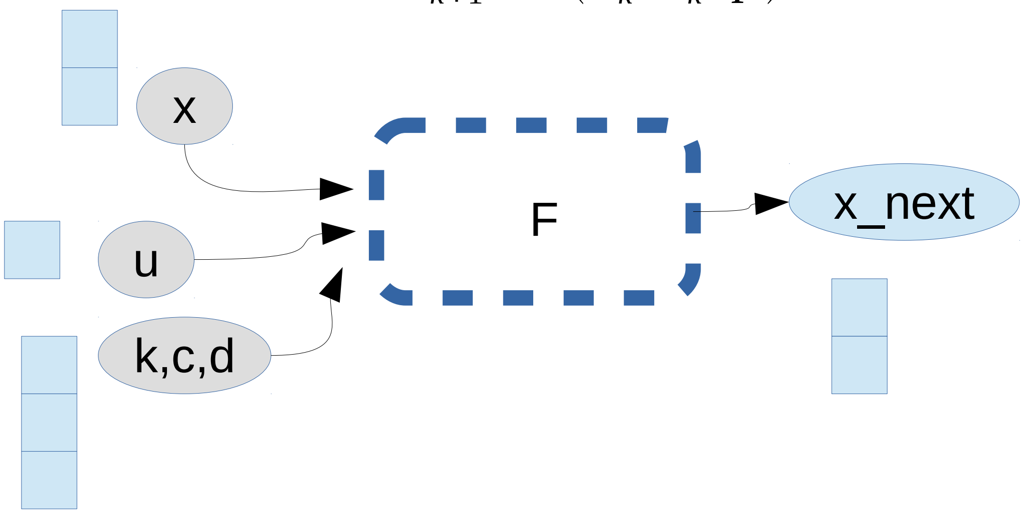


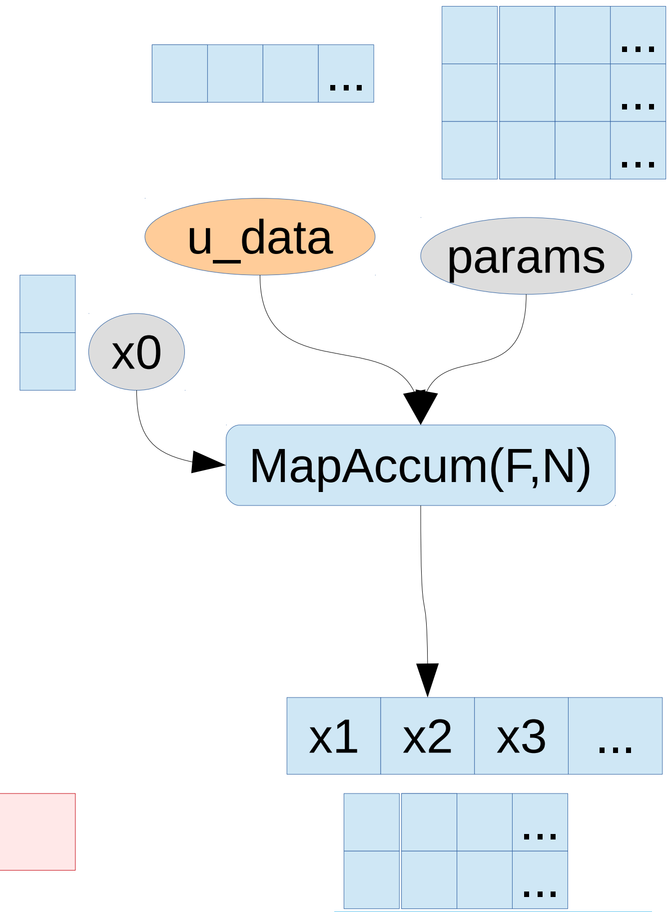
Practical example: sys id

$$\underset{x_0, k, c, d}{\text{minimize}} \quad \left\| x_i - \bar{x}_i \right\|_2^2$$

$$\dot{x} = v$$
$$\dot{v} = u - k\,x - c\,v - d\,x^3$$

$$x_{k+1} = F(x_k, u_k, p)$$






```
Function("F",[x,u,vertcat([k,c,d])],[...])
```

Practical example: sys id

$$\underset{x_0, k, c, d}{\text{minimize}} \quad \left\| x_i - \bar{x}_i \right\|_2^2$$

$$\dot{x} = v$$
$$\dot{v} = u - k\,x - c\,v - d\,x^3$$

Known signals

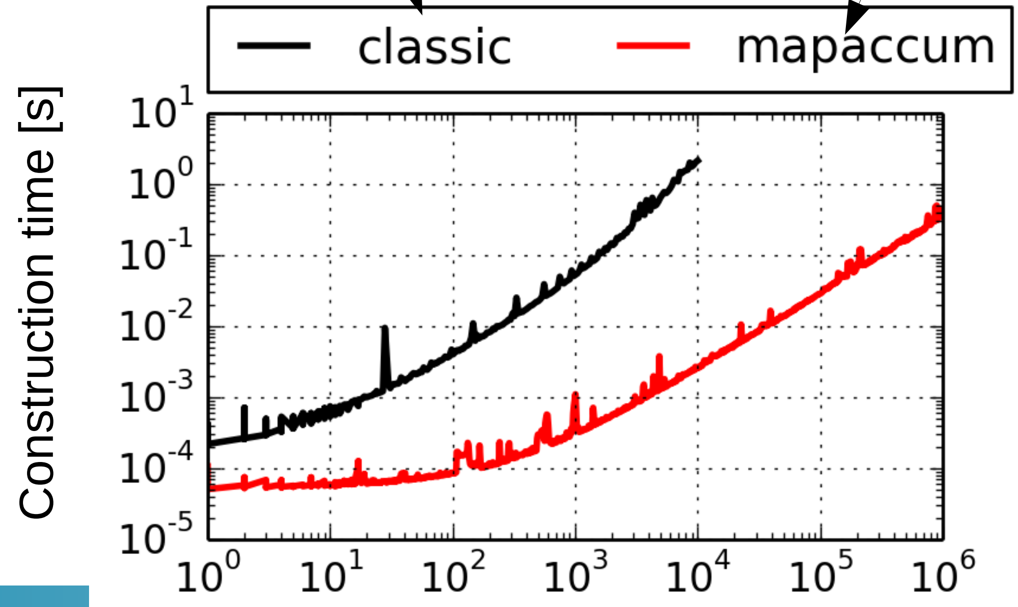
```
R = F.mapaccum("R", N)

X_symbolic = R(x0, u_data, repmat(params,1,N))

e = y_data-X_symbolic[0,:].T;
```

Measured data

Practical example: sys id

```
for i in range(N):
  X = F(X,u_data[:,i],params)
```

```
R = F.mapaccum("R", N)
```



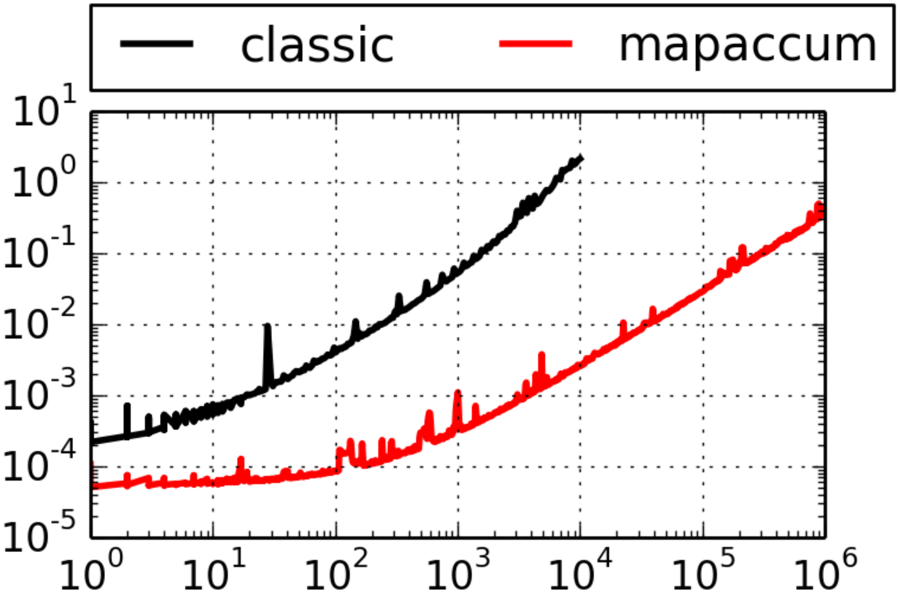
N

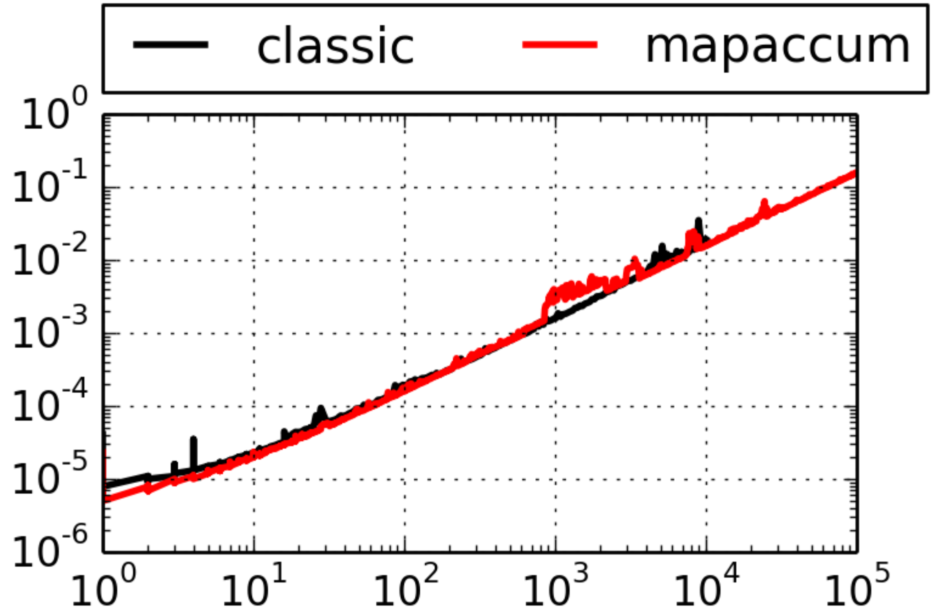
KU LEUVEN

Construction time ↔ evaluation time

Up-front cost

Numerical evaluation
(iterations of nlp)

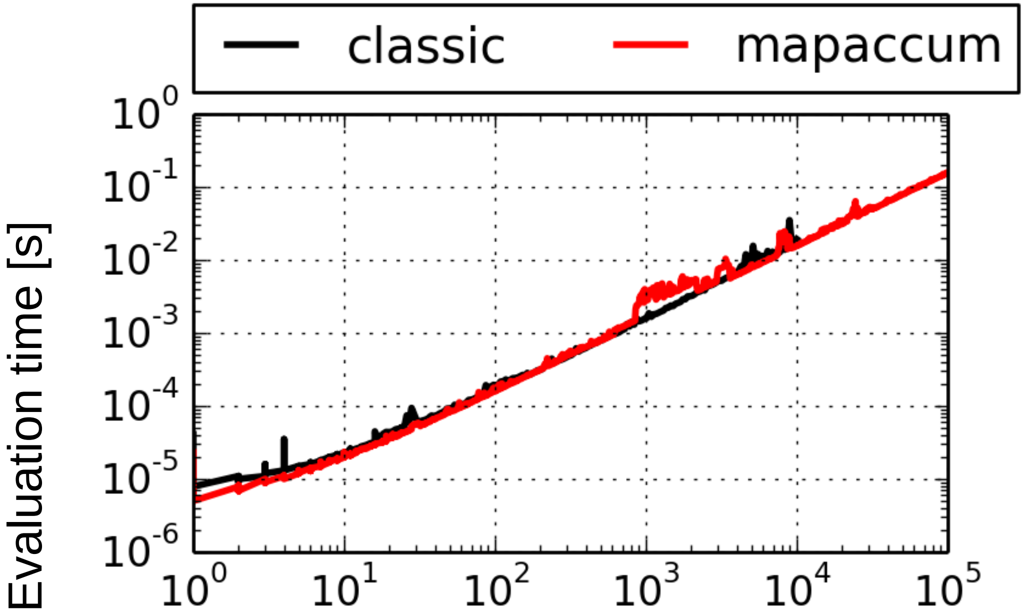




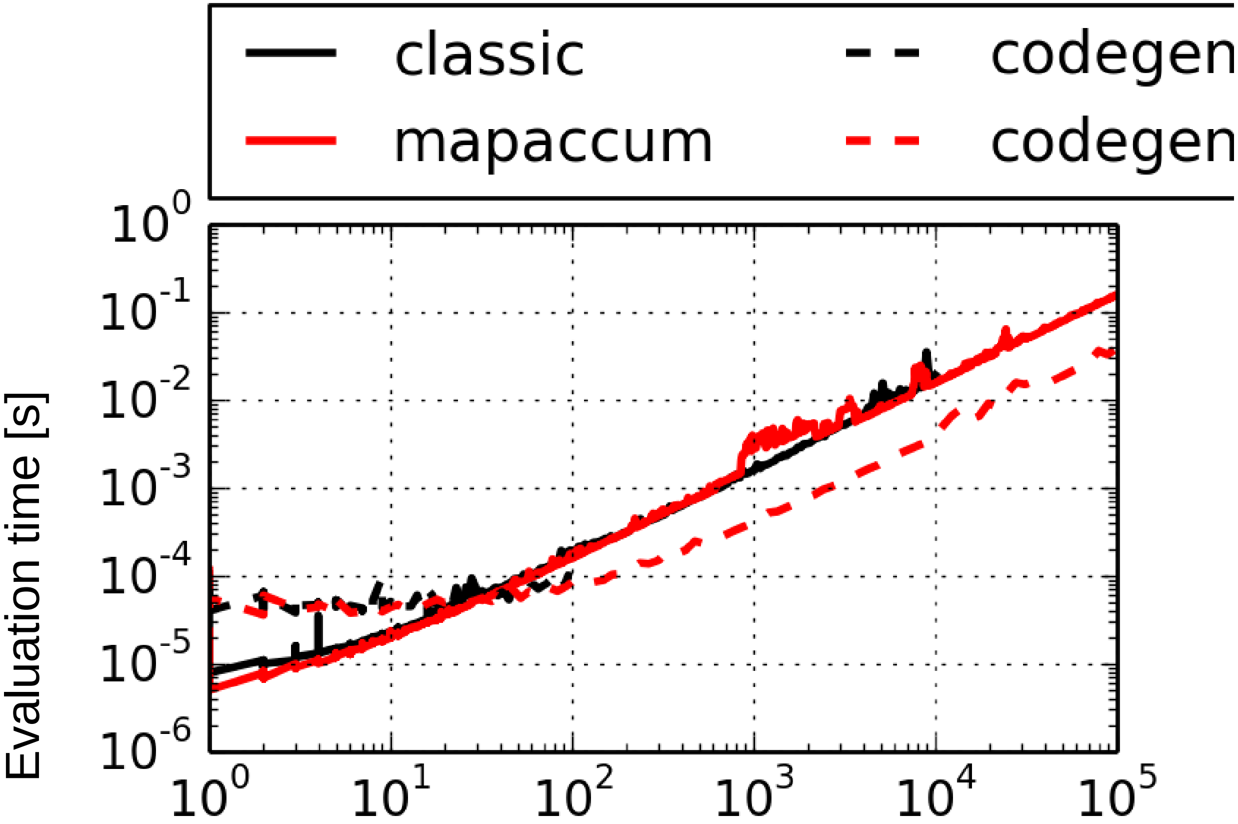
Construction time ↔ evaluation time

Numerical evaluation
(iterations of nlp)

So MapAccum is useless to speed up online computations?



Code-generation

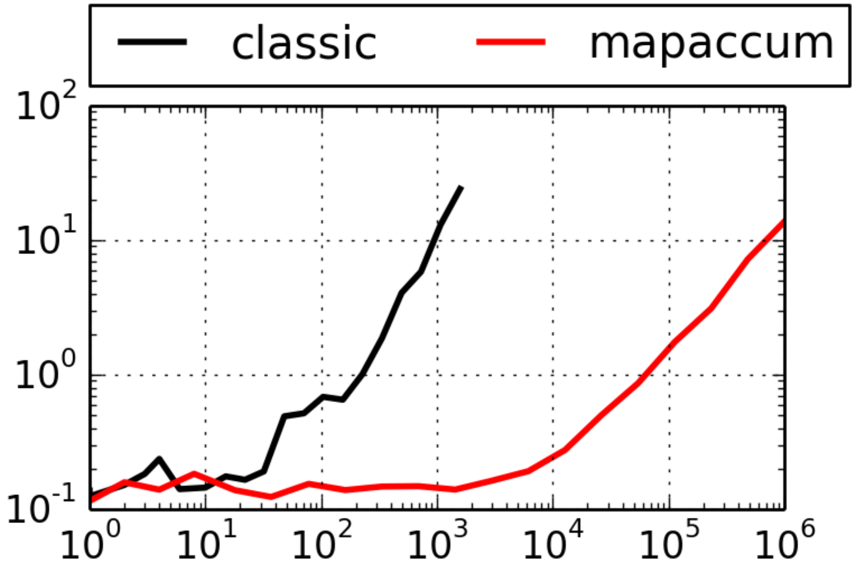


KU LEUVEN

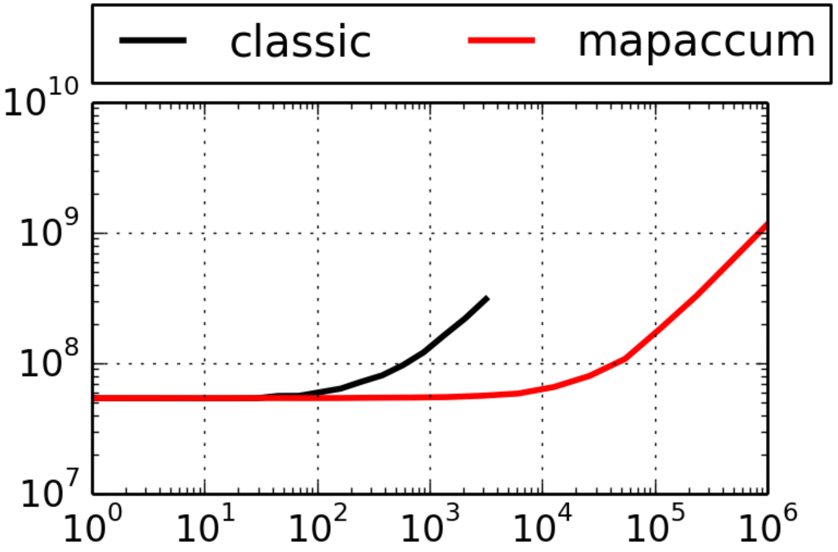
Code-generation

First you need to run the compiler...

Compilation time [s]

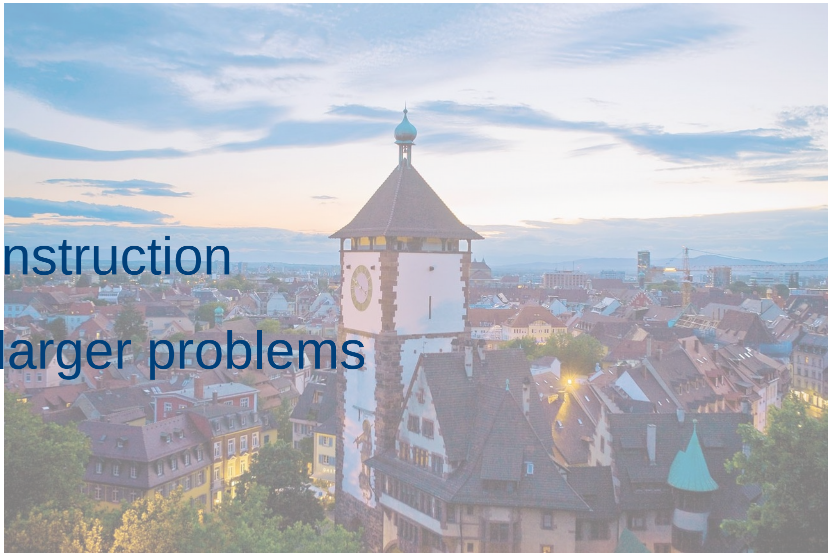


Compilation memory [B]



Outline

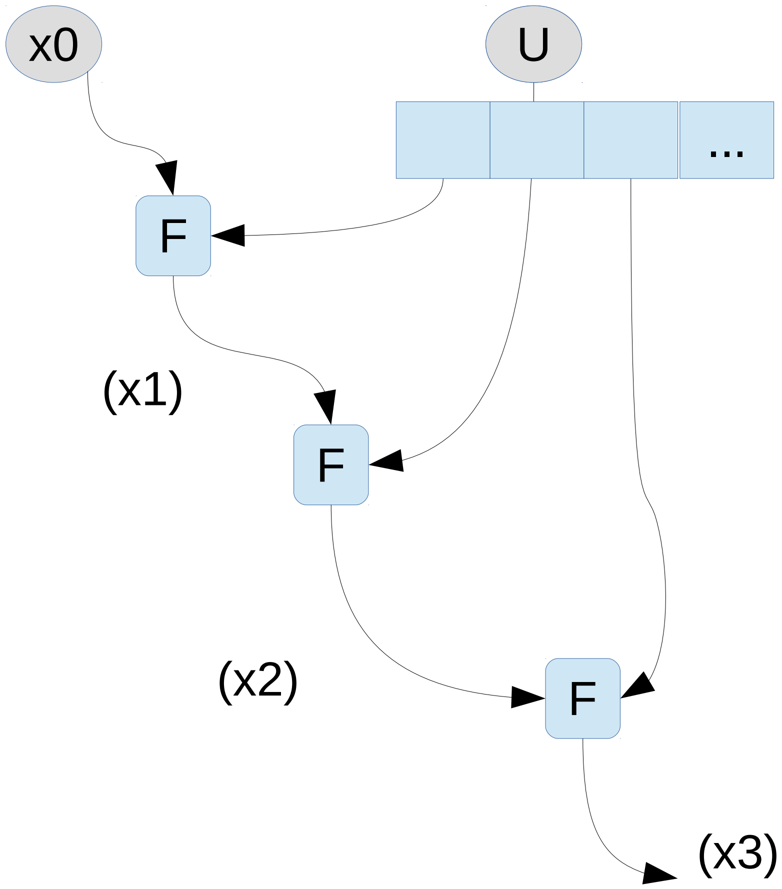
- Key ingredients of CasADi
  - Efficient symbolics
  - Algorithmic differentiation
  - Matrix-valued graphs
  - Function embedding
- New concepts: MapAccum
  - 2 orders of magnitude faster construction
  - compile 2 orders of magnitude larger problems
- Software architecture

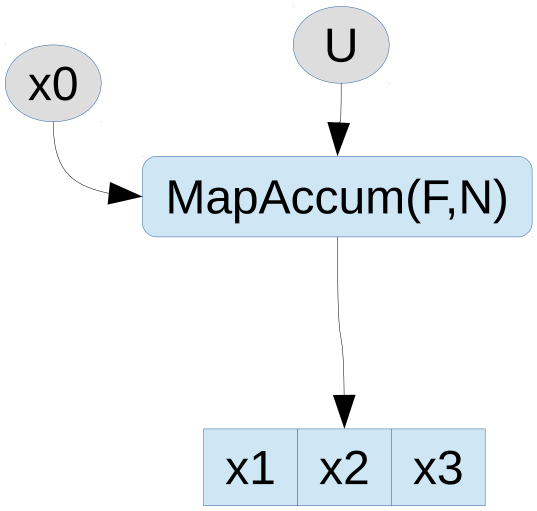


KU LEUVEN

MapAccum ~ single shooting

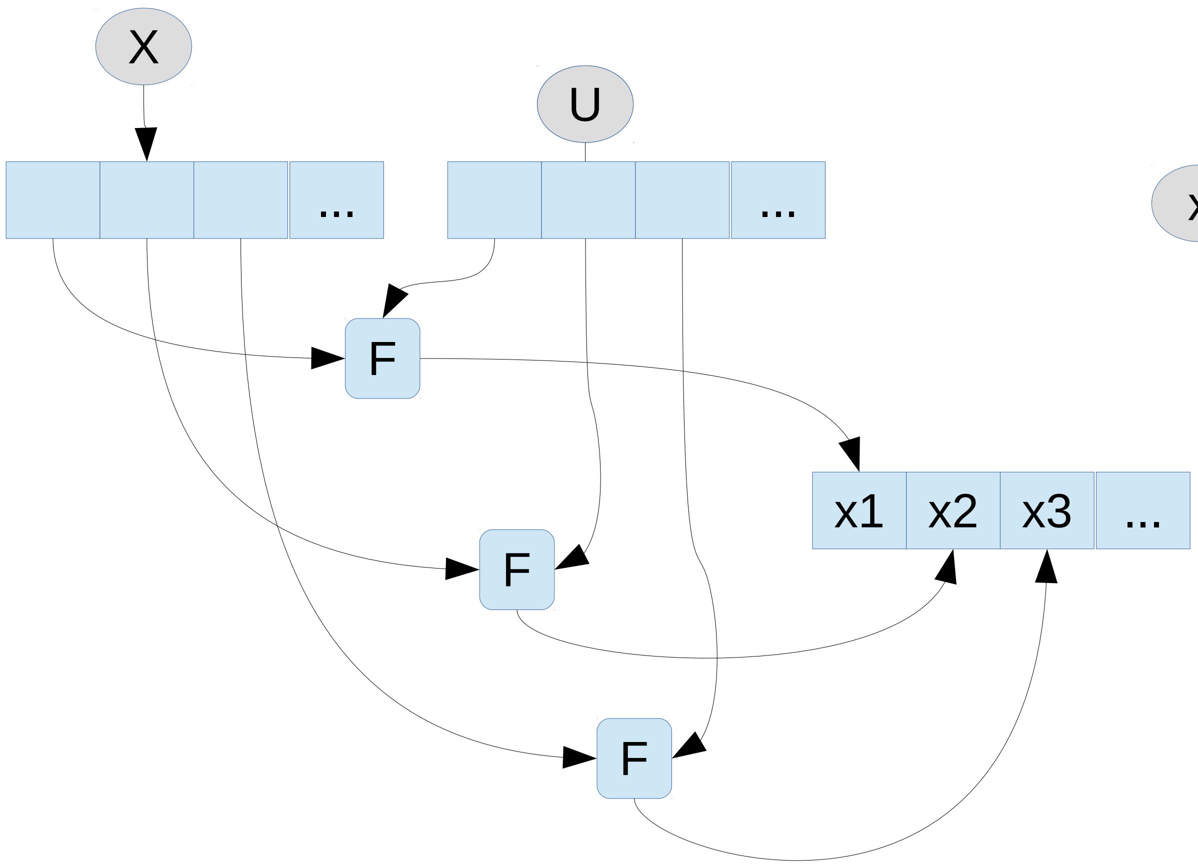
$$\underset{x_0, u_0, u_1 \dots u_{N-1}}{\text{minimize}} \quad x_N^2$$

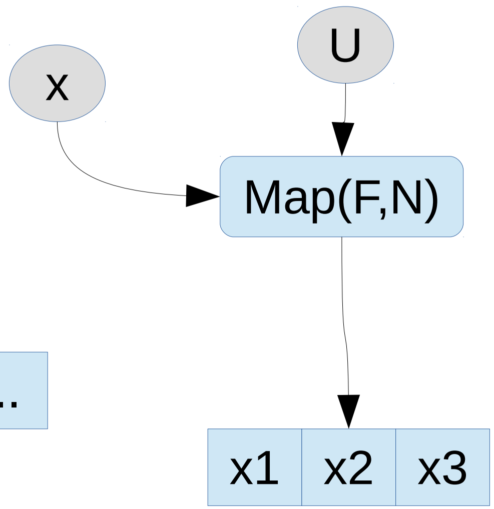




Map ~ multiple shooting

$$\underset{x_{0,}x_{1,}\ldots x_N, u_{0,}u_1\ldots u_{N-1}}{\text{minimize}} \quad x_N^2$$





KU LEUVEN

Map ~ multiple shooting

$$\underset{x_{0,}x_{1,}\dots x_N, u_{0,}u_1\dots u_{N-1}}{\text{minimize}} \quad x_N^2$$

s.t.

$$x_1 - F(x_0; u_0) = 0$$
$$x_2 - F(x_1; u_1) = 0$$
$$\dots$$
$$x_N - F(x_{N-1}; u_{N-1}) = 0$$

F is a function that:
- reads a few inputs
- computes a lot (e.g. time integration)
- writes a few outputs

Ideal for parallelism:
- openmp support builtin
- opencl → GPU (proof-of-concept)

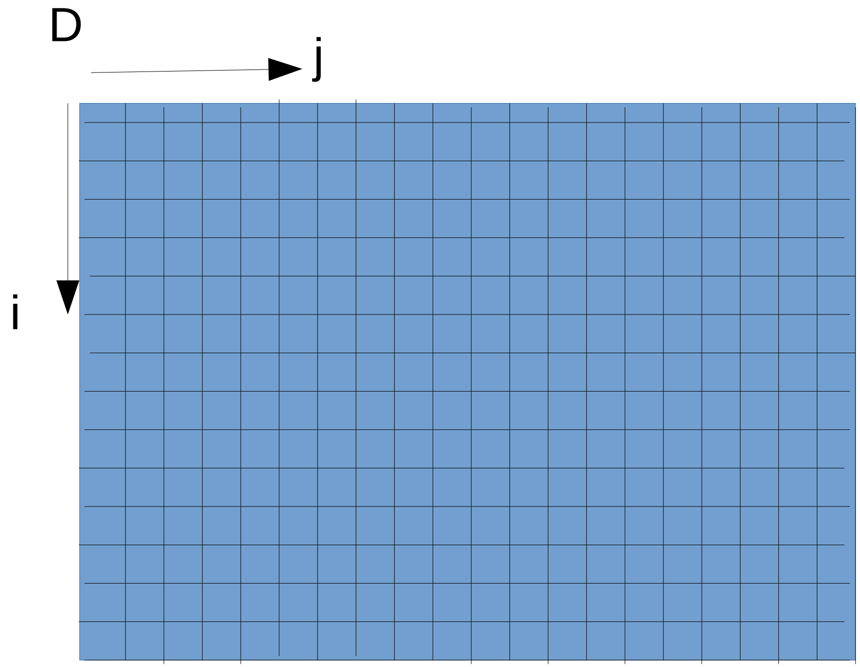
```
R = F.map("R","serial",N)

X_n = R(X, u_data, repmat(params,1,N))

gaps = X[:,1:]-X_n[:,:-1]
```

Outline

- Key ingredients of CasADi
  - Efficient symbolics
  - Algorithmic differentiation
  - Matrix-valued graphs
  - Function embedding
- New concepts: Map, MapAccum
  - Speedups
- Massive parallelization
- Software architecture

Operation on an image

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$



Operation on an image

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$

```
X = MX.sym(“x”,2)
D = MX.sym(“D”,1024,768)

R = F.map(“mymap”,”serial”,1024*768)

Dflat = vec(D).T

terms = R(Dflat,X)

e = sum2(terms)
```

Operation on an image

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$

```
X = MX.sym("x",2)
D = MX.sym("D",1024,768)

R = F.map("mymap","openmp",1024*768)

Dflat = vec(D).T

terms = R(Dflat,X)

e = sum2(terms)
```

Operation on an image

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$

```
X = MX.sym("x",2)
D = MX.sym("D",1024,768)

R = F.map("mymap","opencl",1024*768)

Dflat = vec(D).T

terms = R(Dflat,X)

e = sum2(terms)
```

KU LEUVEN

Operation on an image

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$

```
X = MX.sym(“x”,2)
D = MX.sym(“D”,1024,768)

R = F.map(“mymap”,”opencl”,1024*768)

Dflat = vec(D).T

terms = R(Dflat,X)

e = sum2(terms)
```

What is OpenCL?

- C library
- Compiler for computation kernels
- Low-level memory concepts
- Write once, run on
  - CPU
  - GPU
  - FPGA

Operation on an image

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$

```
X = MX.sym("x",2)
D = MX.sym("D",1024,768
R = F.map("mymap","opencl
Dflat = vec(D).T
terms = R(Dflat,X)
e = sum2(terms)
```

Init:
Allocate buffer for x on GPU
Allocate buffer for D on GPU
Allocate buffer for result on GPU
Compile GPU kernel

Eval:
Send x
Send D
Compute kernel
Retrieve sol

KU LEUVEN

Kernelsum node

$$\sum_{i,j} f\left(D_{i,j}, X\right)$$

Init:
Allocate buffer for x on GPU
Allocate buffer for D on GPU
Allocate buffer for result on GPU
Send D
Compile GPU kernel

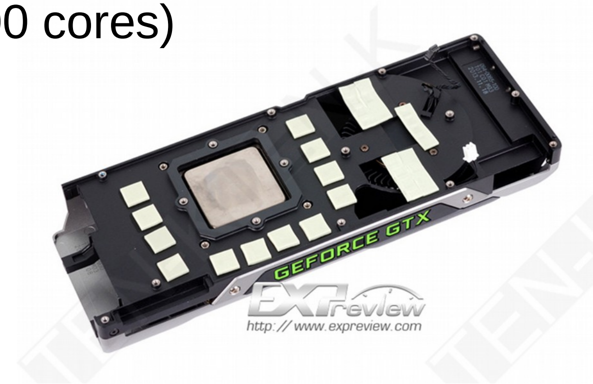
Eval:
Send x
Compute kernel
Sum result
Retrieve sum

KU LEUVEN

Kernelsum node

For Flanders Make:

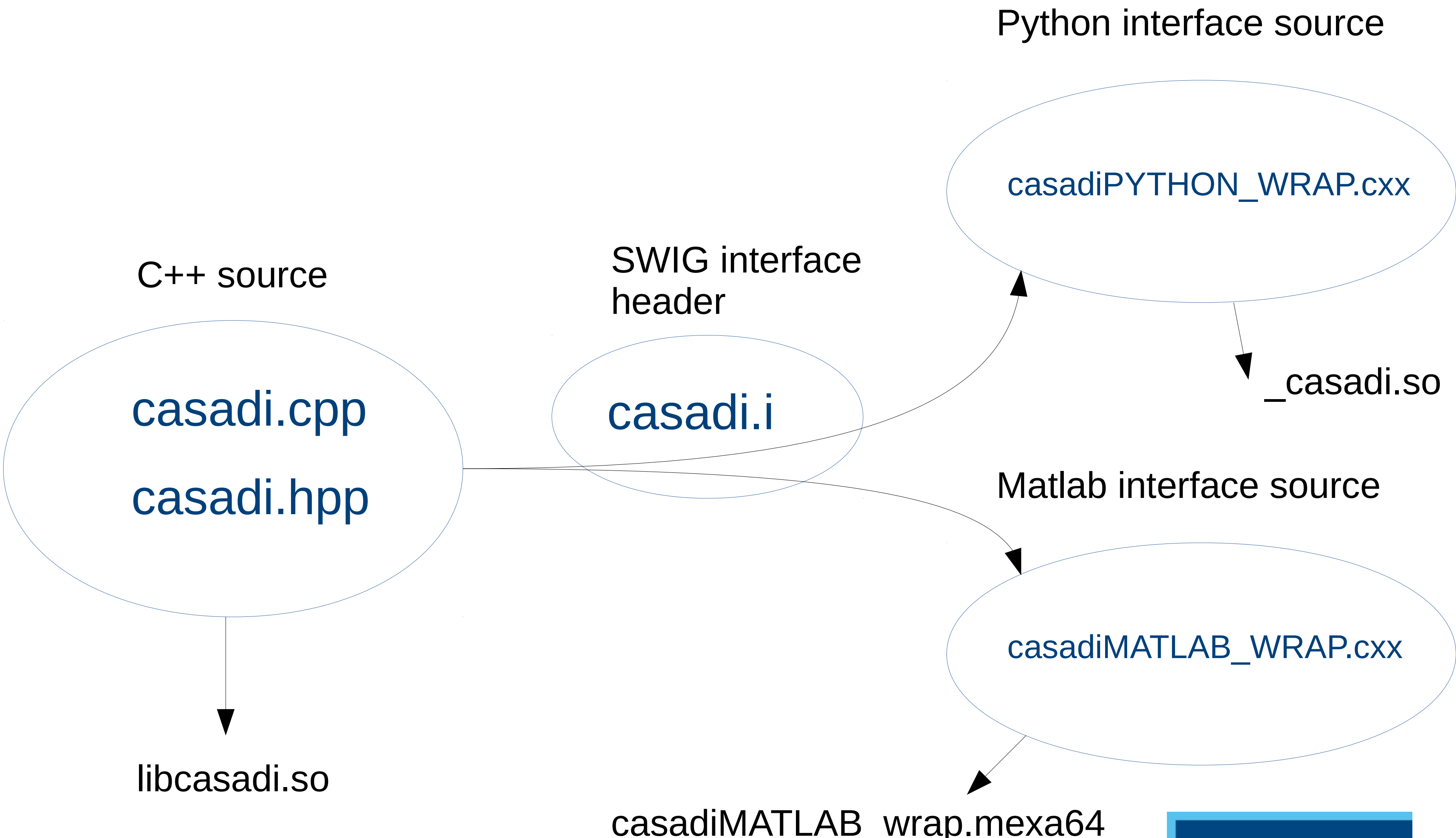
100..1000 x speedup on a Titan Black GPU (2000 cores)



Outline

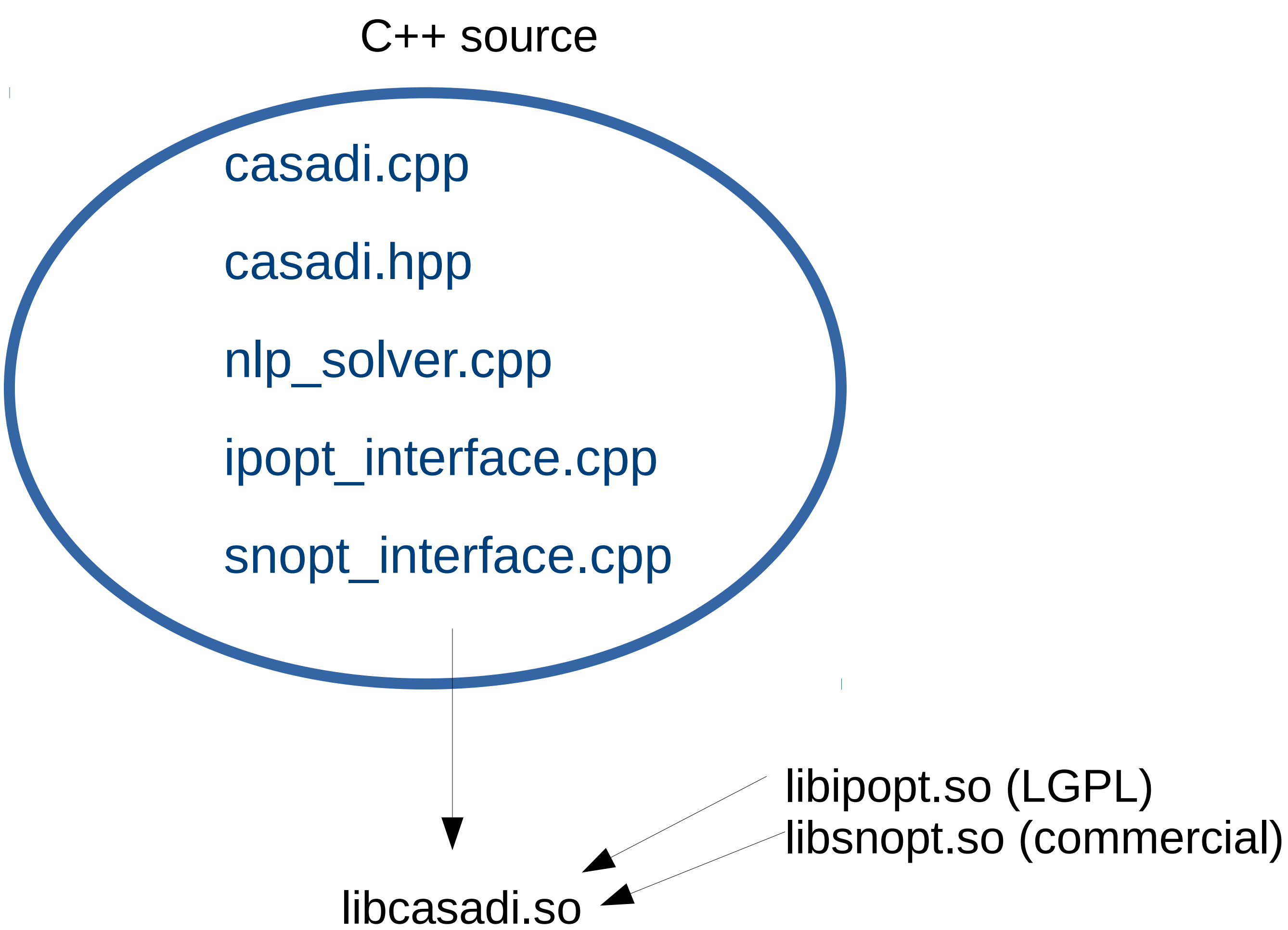
- Key ingredients of CasADi
  - Efficient symbolics
  - Algorithmic differentiation
  - Matrix-valued graphs
  - Function embedding
- New concepts: Map, MapAccum
- Massive parallelization
- Software architecture

Software architecture: interfaces



KU LEUVEN

Software architecture: before plugins



Software architecture: before plugins

C++ source

- casadi.cpp
- casadi.hpp
- nlp_solver.cpp
- ipopt_interface.cpp
- snopt_interface.cpp

libcasadi.so

libipopt.so (LGPL)
libsnopt.so (commercial)

Mr. Cheapskate

```
from casadi import *
```

Error: symbol snopt not found

Mrs. Spender

```
from casadi import *

nlpsol(“solver“,”snopt”,nlp)
```

KU LEUVEN

Software architecture: before plugins

C++ source

- casadi.cpp
- casadi.hpp
- nlp_solver.cpp
- ipopt_interface.cpp
- snopt_interface.cpp

↓

libcasadi.so ← libipopt.so (LGPL), libsnopt.so (commercial)

Mr. Cheapskate

```
from casadi import *
nlpsol(“solver”,“ipopt”,nlp)
```

Mrs. Spender

```
from casadi import *
nlpsol(“solver”,“snopt”,nlp)
```

Error: solver snopt not compiled

Software architecture: plugins

C++ source

- casadi.cpp
- casadi.hpp
- nlp_solver.cpp
- ipopt_interface.cpp
- snopt_interface.cpp

libcasadi.so

```
from casadi import *

nlpsol(“ipopt”,nlp)
```

Dynamically load (dlopen)
libcasadi_nlpsolver_ipopt.so

```
from casadi import *

nlpsol(“snopt”,nlp)
```

Dynamically load (dlopen)
libcasadi_nlpsolver_snopt.so

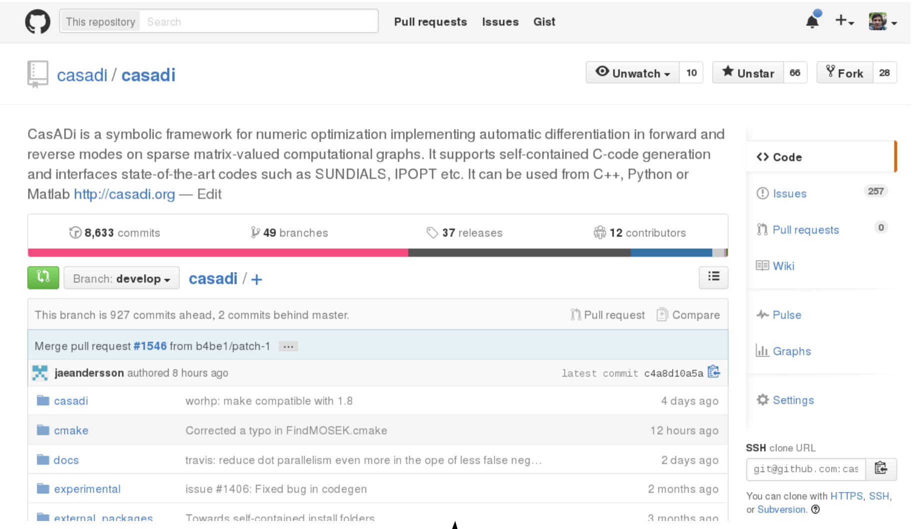
libcasadi_nlpsol_ipopt.so ← libipopt.so (LGPL)
libcasadi_nlpsol_snopt.so ← libsnopt.so (commercial)

KU LEUVEN

Software architecture: version control

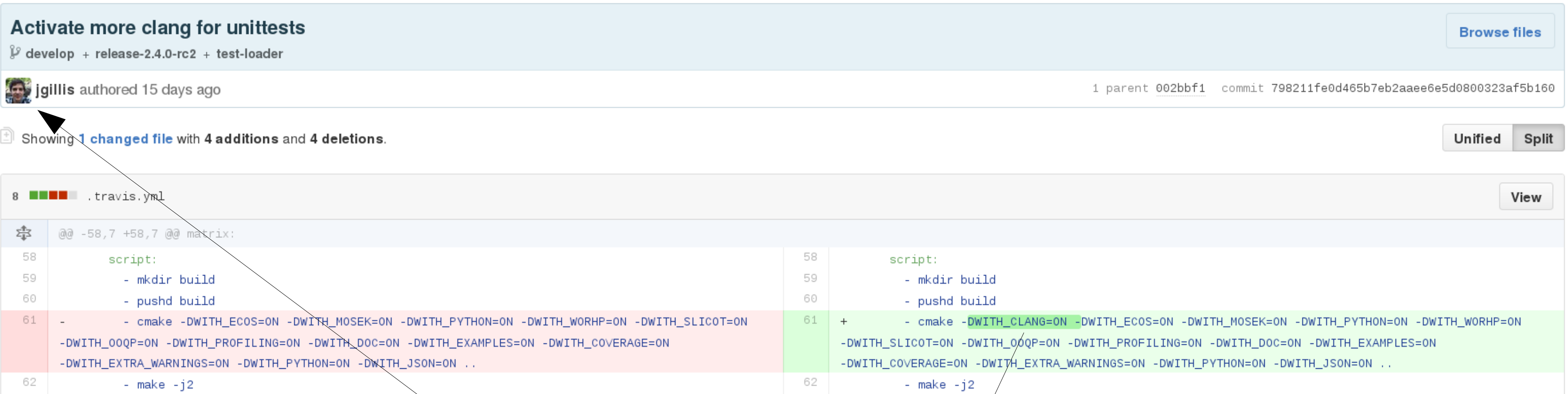
C++ source

- casadi.cpp
- casadi.hpp
- nlp_solver.cpp
- ipopt_interface.cpp
- snopt_interface.cpp



Github

Software architecture: version control

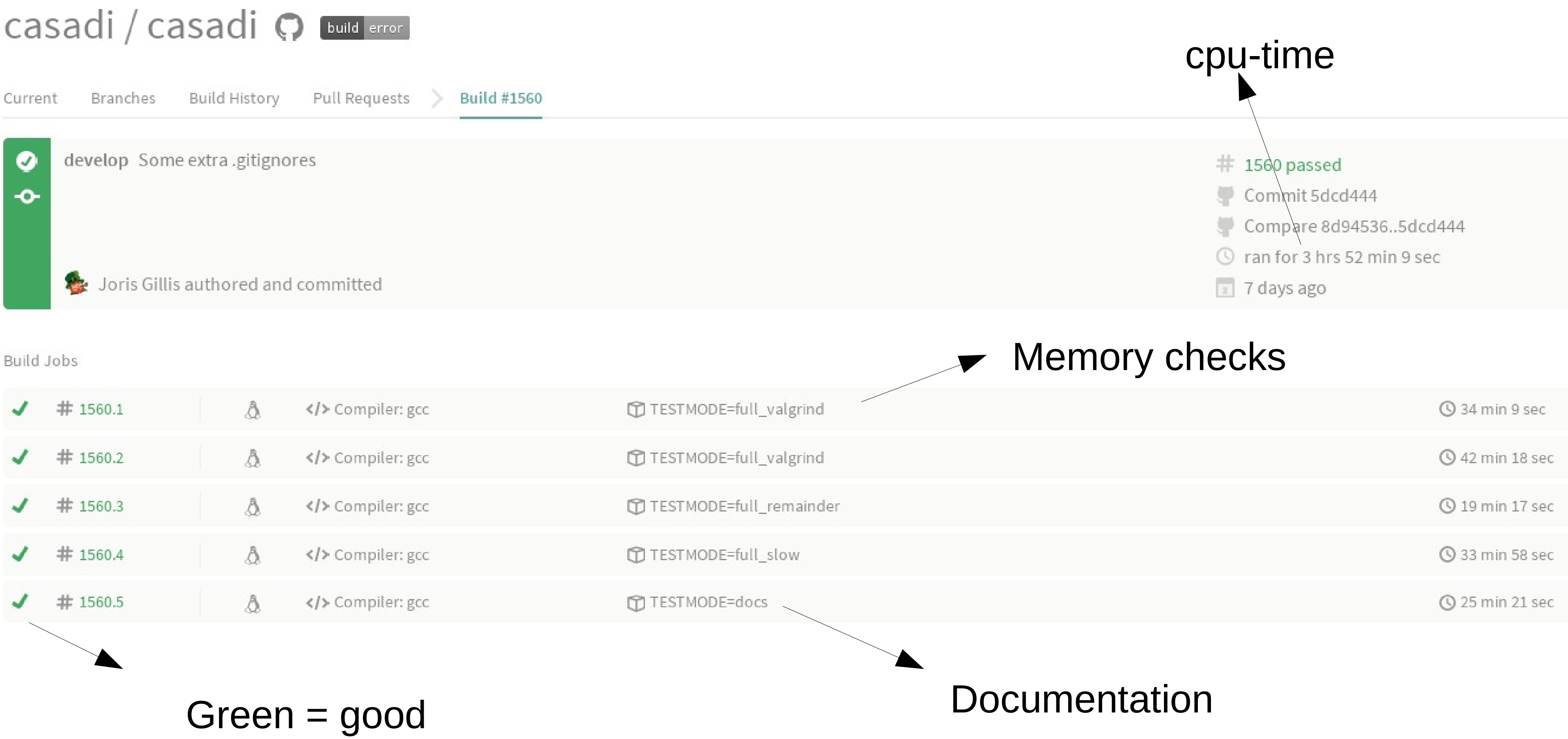


I added a flag here

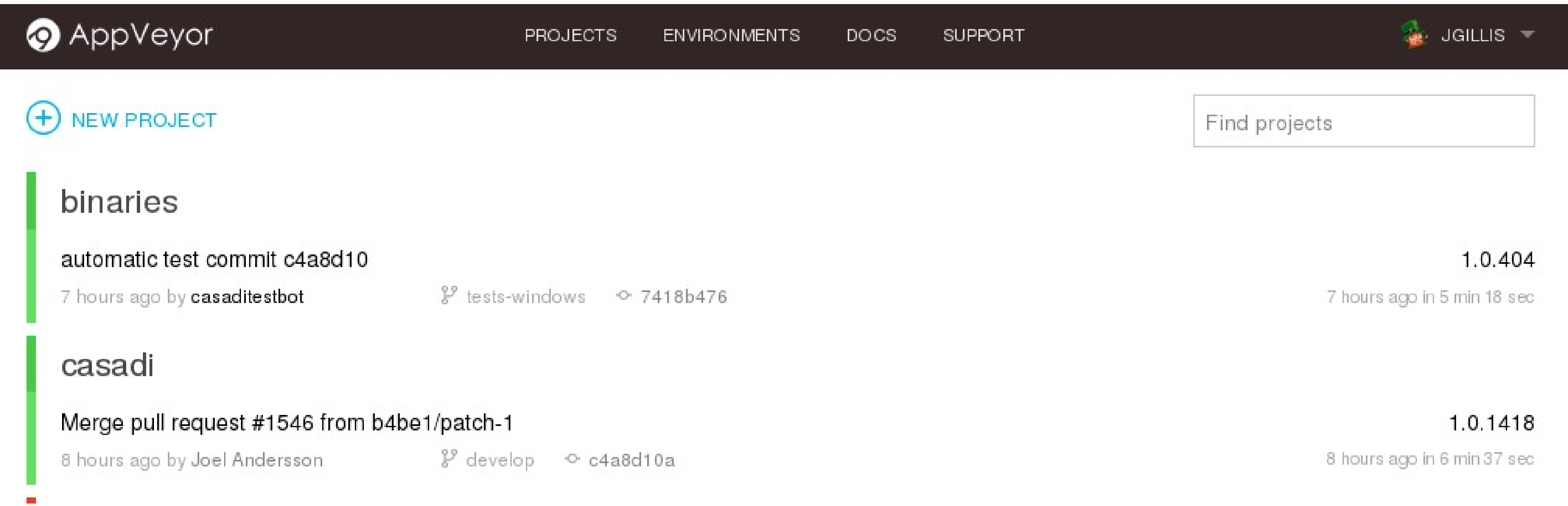
KU LEUVEN

Software architecture: continuous-integration

Every change (commit) is unittested by travis-ci



Software architecture: continuous-integration



KU LEUVEN

Software architecture: continuous-integration

Both appveyor and travis are free for open-source projects

Encryption support → can test commercial plugins/interfaces

e.g. Matlab

Software architecture: binaries

Linux buildslaves

g++
x86_64-w64-mingw32-g++

Home / CasADi / commits / 2e60be1

Name	Modified	Size	Downloads / Week
Parent folder			
linux	2015-09-04		18
windows	2015-09-04		30
osx	2015-09-04		7
casadi-docs-2e60be1.zip	2015-09-05	59.2 MB	6
casadi-example_pack-2e60be1.zip	2015-09-05	603.7 kB	7
README.md	2015-09-05	81 Bytes	6
Totals: 6 Items		59.8 MB	19

Software architecture: binaries (python)

Binary = zip folder

e.g. casadi-py27-np1.9.1-v3.0.0.tar.gz

/home/me/software
- casadi-py27-np1.9.1-v2.4.0
  - casadi
  - include
  - lib
- casadi-py27-np1.9.1-v3.0.0
  - casadi
  - include
  - lib

```
import sys
sys.path.append("/home/me/software/casadi-py27-np1.9.1-v3.0.0")

import casadi
```

Software architecture: binaries (matlab)

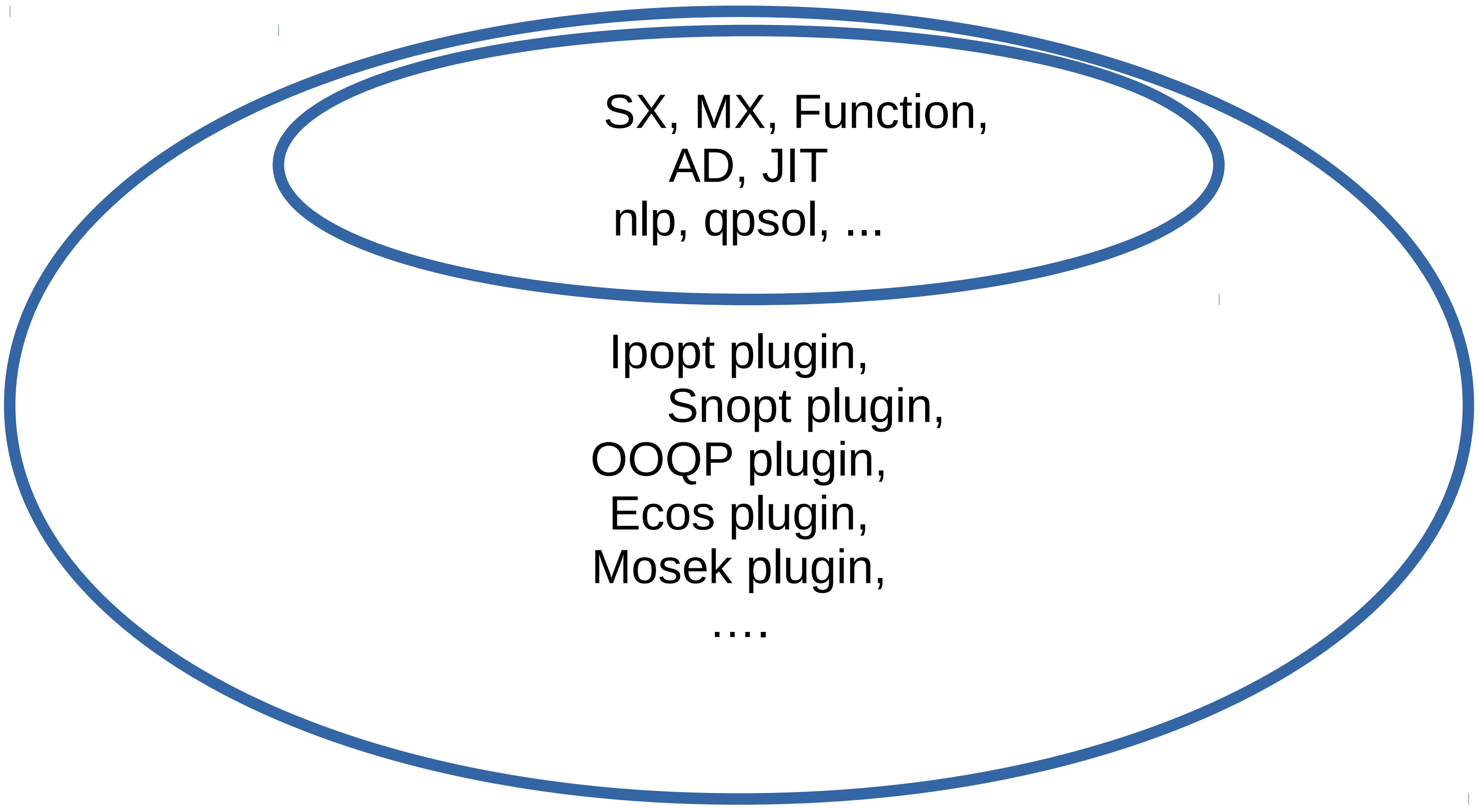
Binary = zip folder

e.g. casadi-matlabR2014b-v2.4.0.zip

```
/home/me/software
  - casadi-matlabR2014b-v2.4.0
     - +casadi
     - include
     - lib
  - casadi-matlabR2014b-v3.0.0
     - +casadi
     - include
     - lib
```

```
addpath('/home/me/software/casadi-matlabR2014b-v3.0.0')

import casadi.*
```

CasADi



Developers: Joel Andersson, Joris Gillis
Greg Horn, Niels van Duijkeren

KU LEUVEN

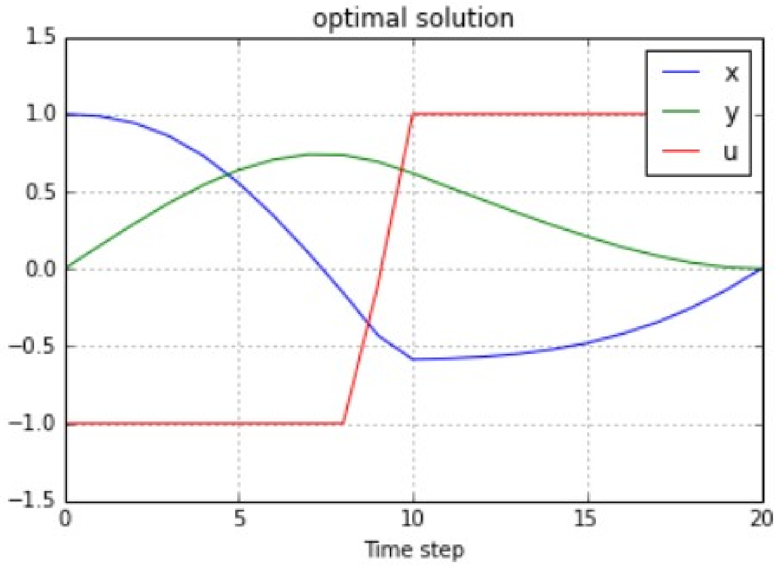
CasADi

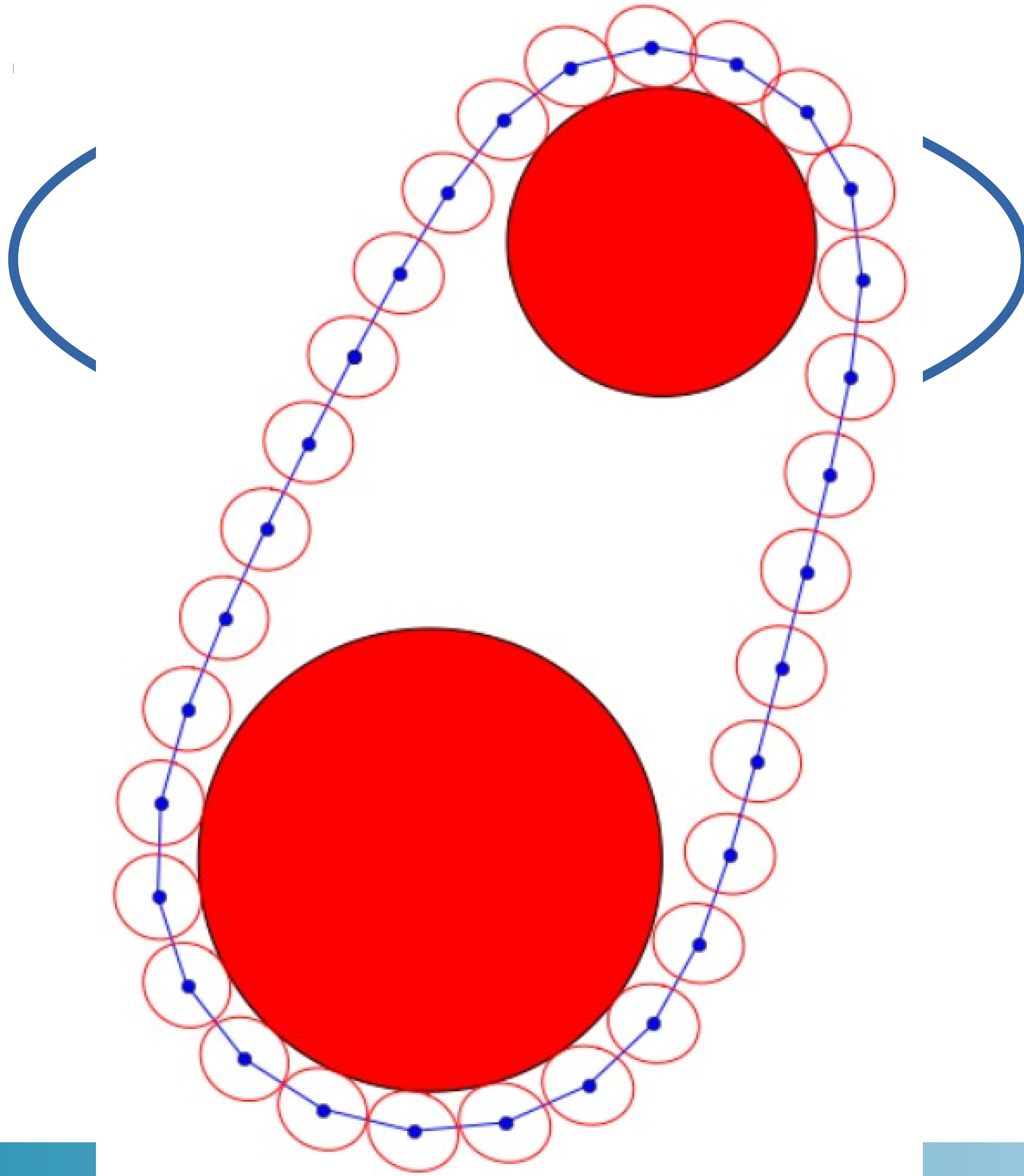
Go write your dynamic optimization tool...

KU LEUVEN

e.g. Optoy

```
from optoy import *

x = state()
y = state()
q = state()
u = control()

T = var(lb=0,init=10)

x.dot = (1-y**2)*x-y+u
y.dot = x
q.dot = x**2+y**2

ocp(T,[u>=-1,u<=1,q.start==0,x.start==1,y.start==0,x.end==0,y.end==0],T=T,N=20)

plot(x.sol)
plot(y.sol)
plot(u.sol)
```





KU LEUVEN

CasADi

optistack

A simple matlab interface to casadi

The goal of this project is to provide a Yalmip-like Matlab interface to casadi.

```
x = optivar();
y = optivar();

nlp = optisolve((1-x)^2+100*(y-x^2)^2,{x^2+y^2<=1, x+y>=0});

optival(x)
optival(y)
```

CasADi

Go write your dynamic optimization tool...

Prototype-style speed of development
State-of-the-art performance

KU LEUVEN

Thanks for your attention

Let's do some coding...

KU LEUVEN