

Lecture 5

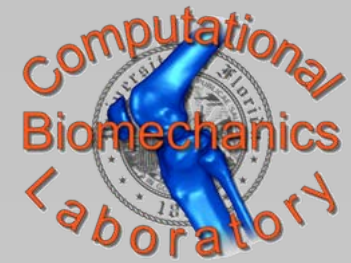
Dynamic Modeling with SD/Fast

EML 5595

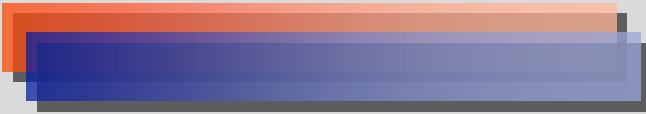
Mechanics of the Human Locomotor System



Outline



- Motivation for Computational Tools
- Big Picture of Computational Tools
- Overview of SD/Fast

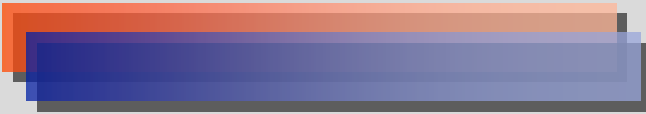




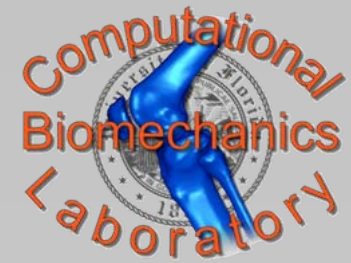
Outline



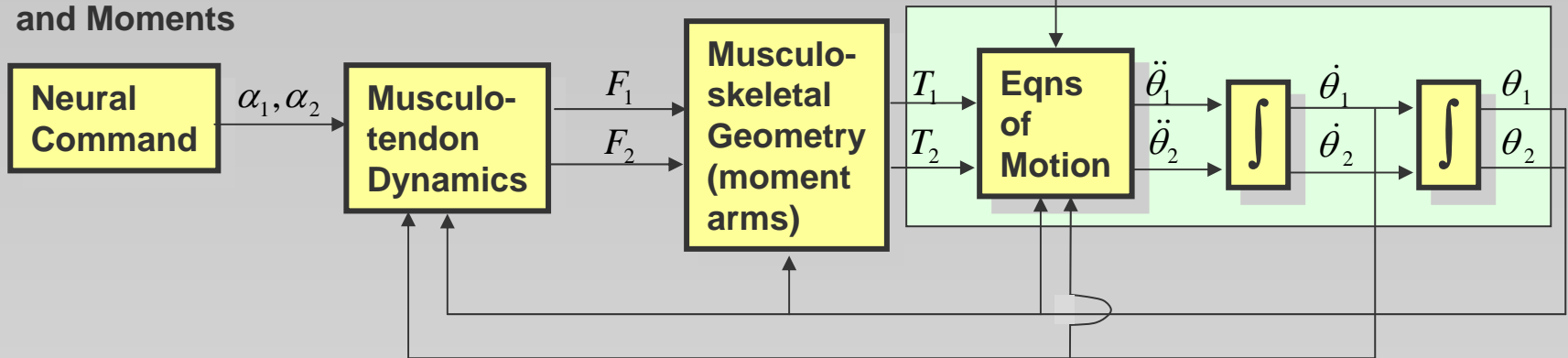
- Motivation for Computational Tools



Course Outline

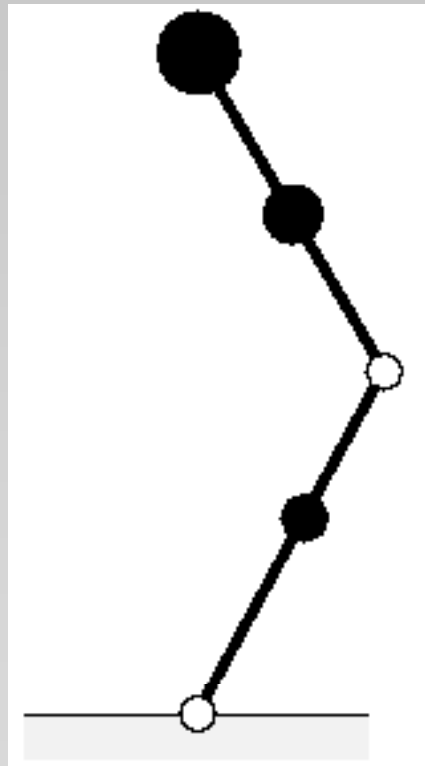
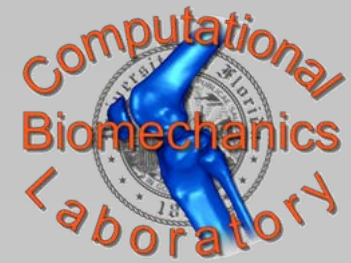


External Forces
and Moments



1. Systems-level Modeling
2. Multi-joint Dynamics

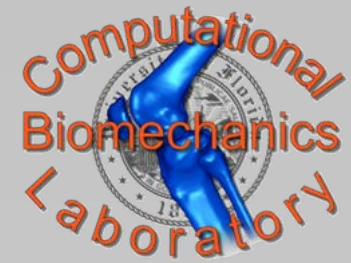
What Is It?



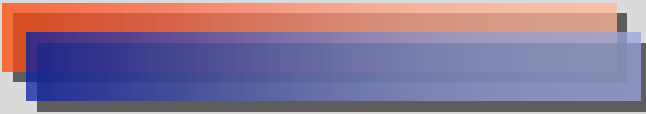
Motivation



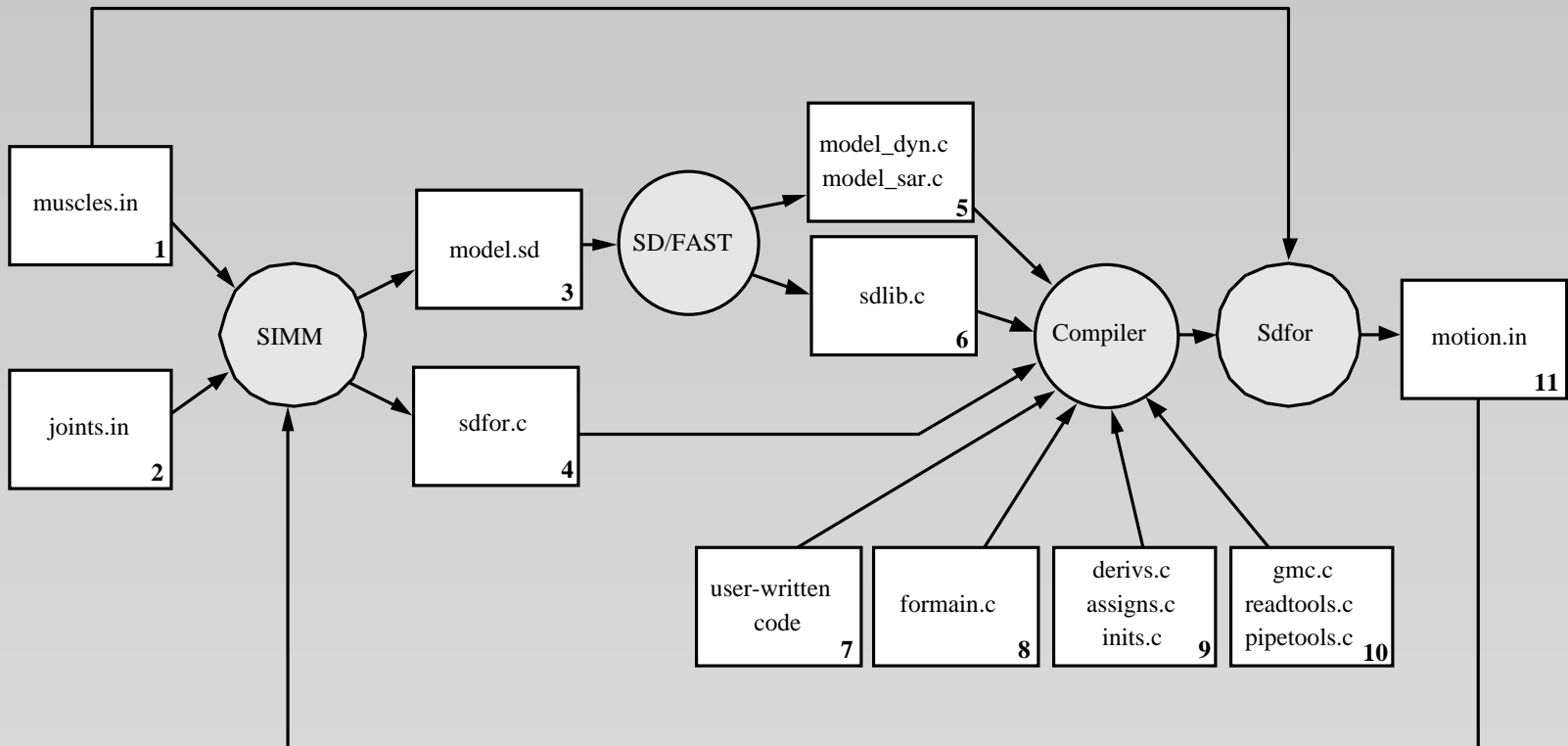
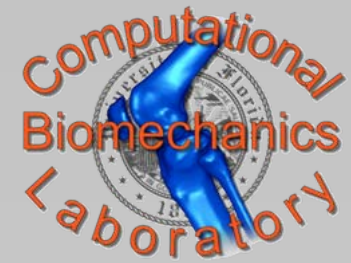
Outline



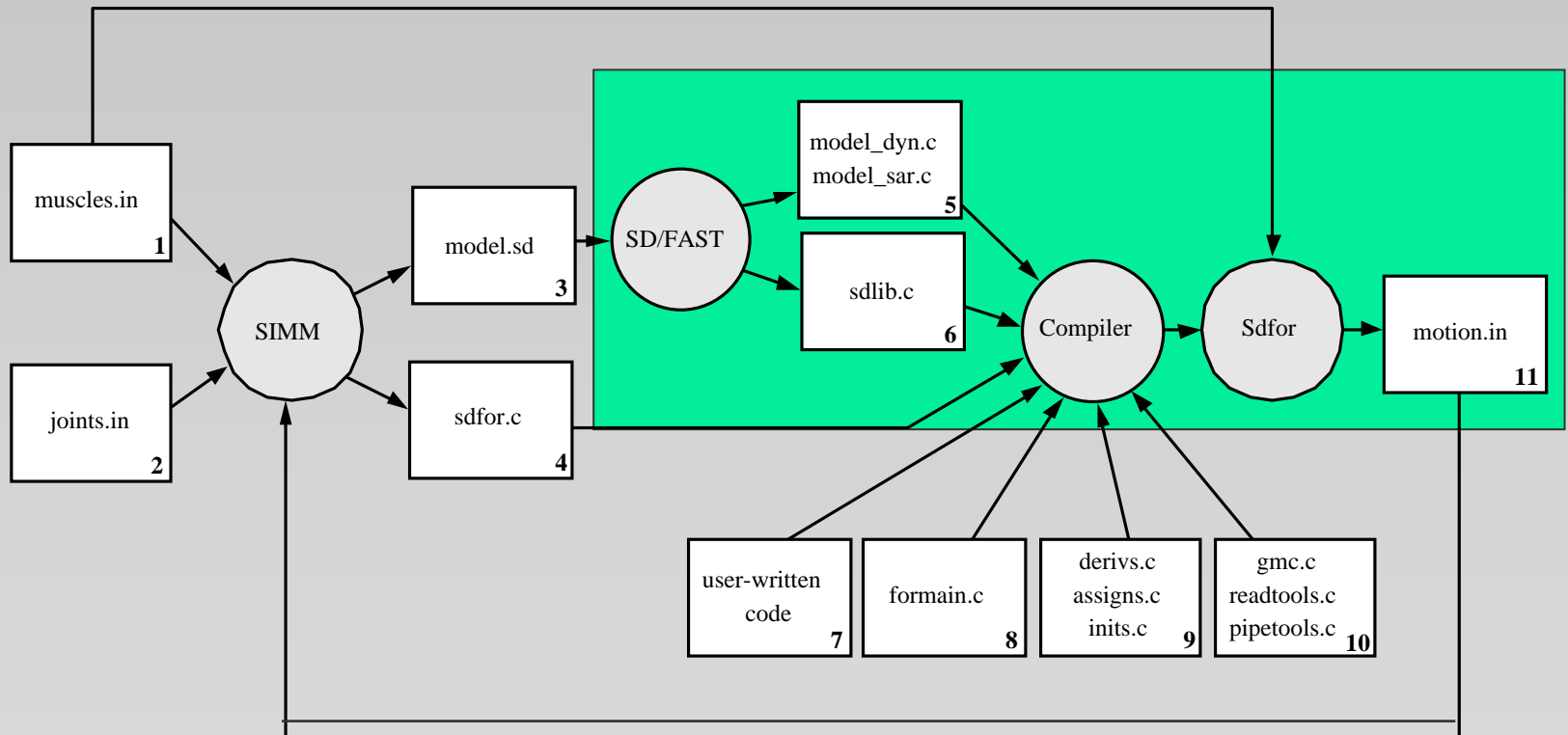
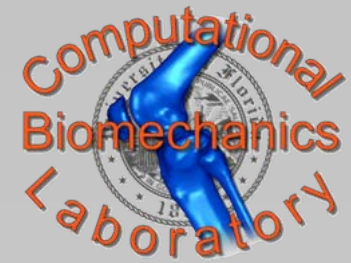
- Motivation for Computational Tools
- **Big Picture of Computational Tools**



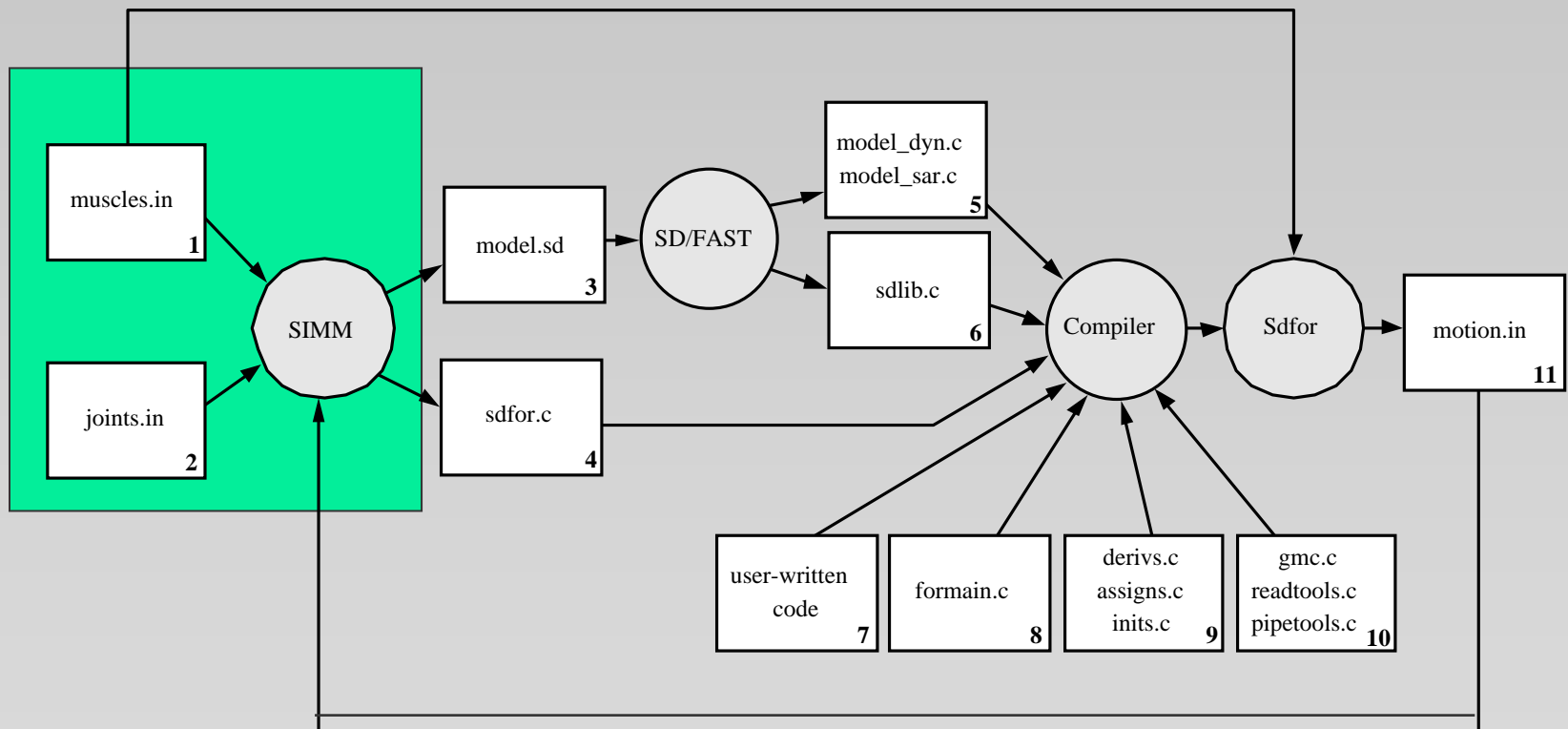
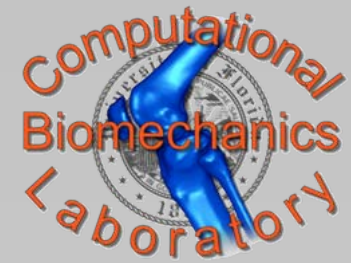
Computational Tools



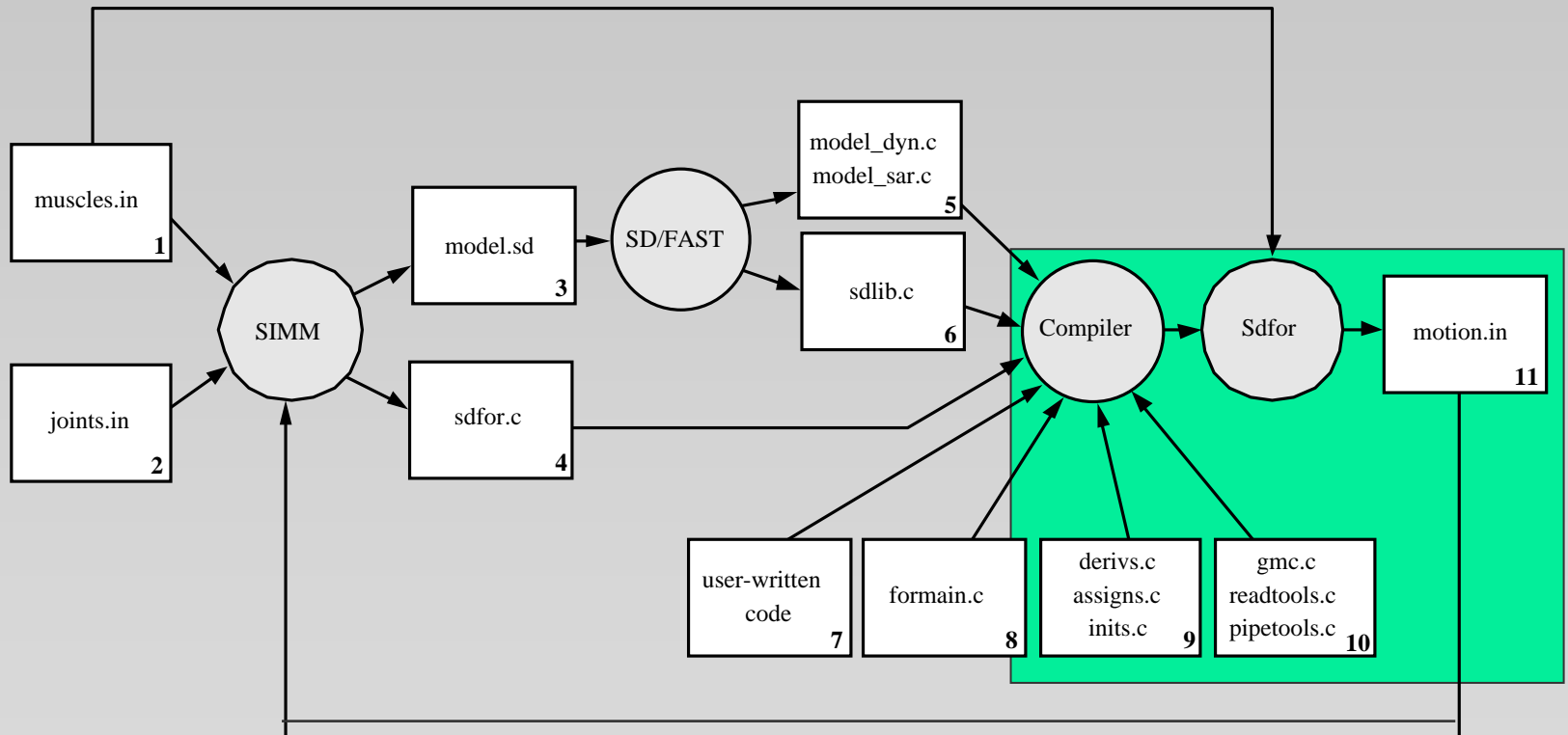
Lab 2



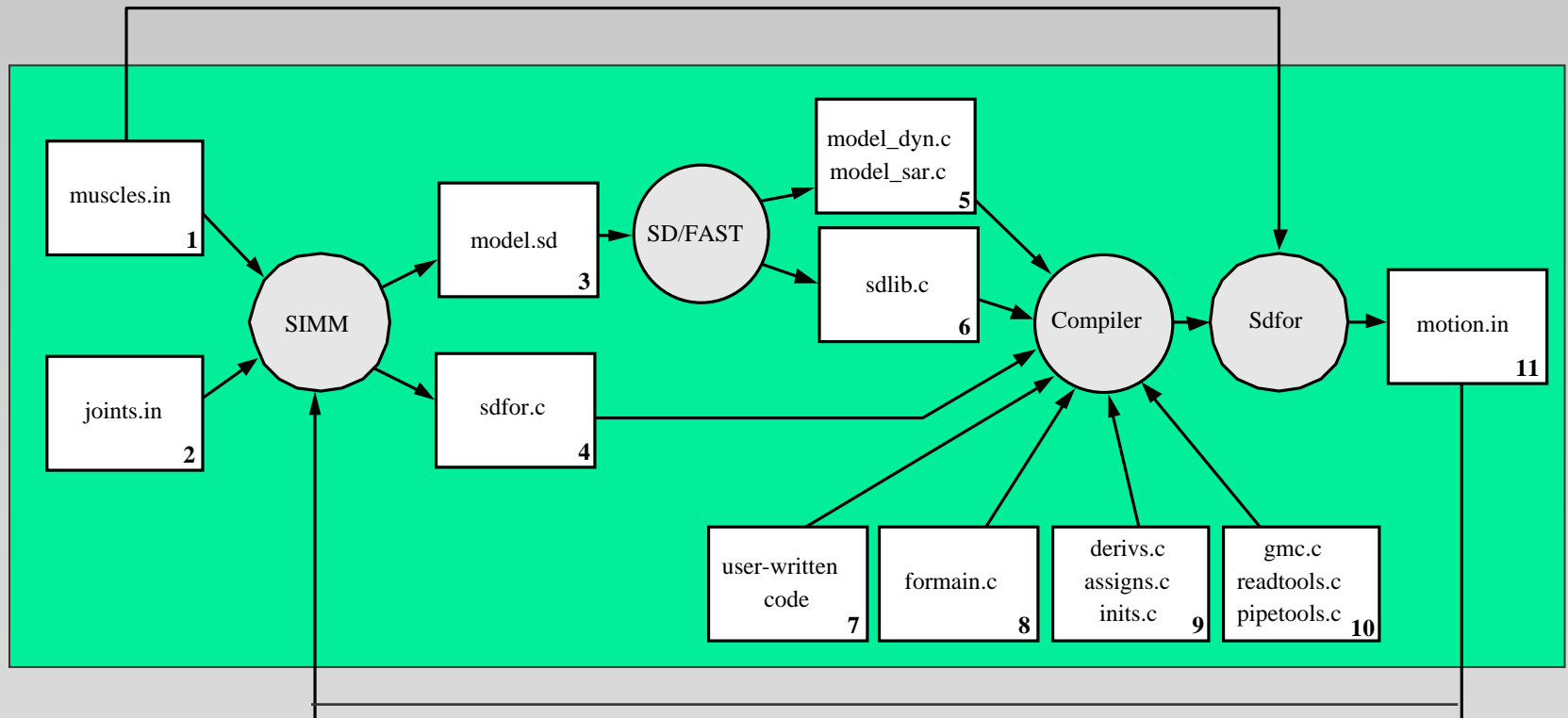
Lab 3



Lab 4

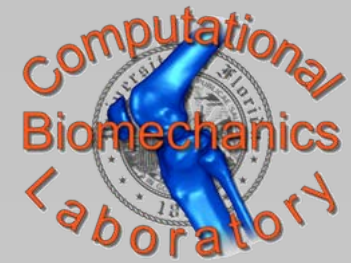


Lab 5

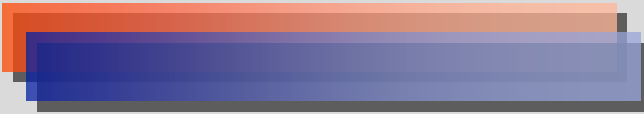




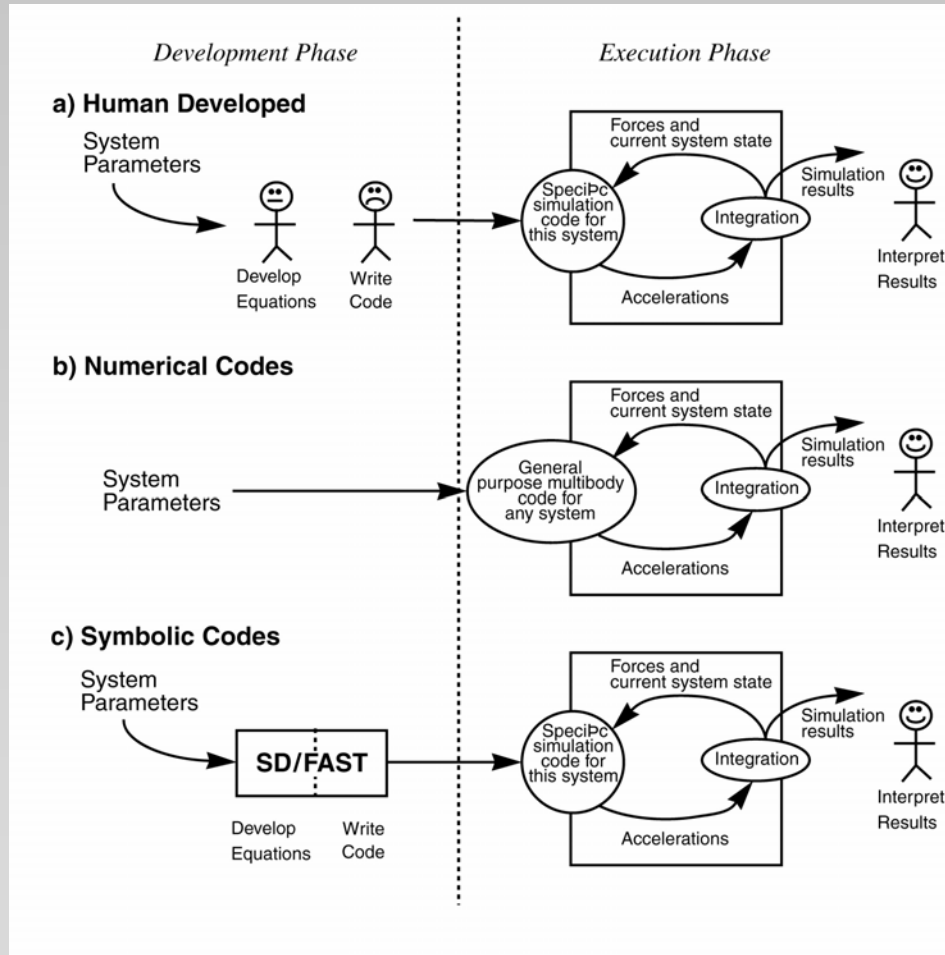
Outline



- Motivation for Computational Tools
- Big Picture of Computational Tools
- **Overview of SD/Fast**



Dynamic Simulation Methods



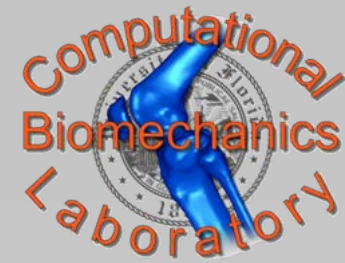
Adams
Working Model
DADS
Mechanica Motion

SD/FAST
AutoLev

Advantages:

- More efficient
- Access to equations and code

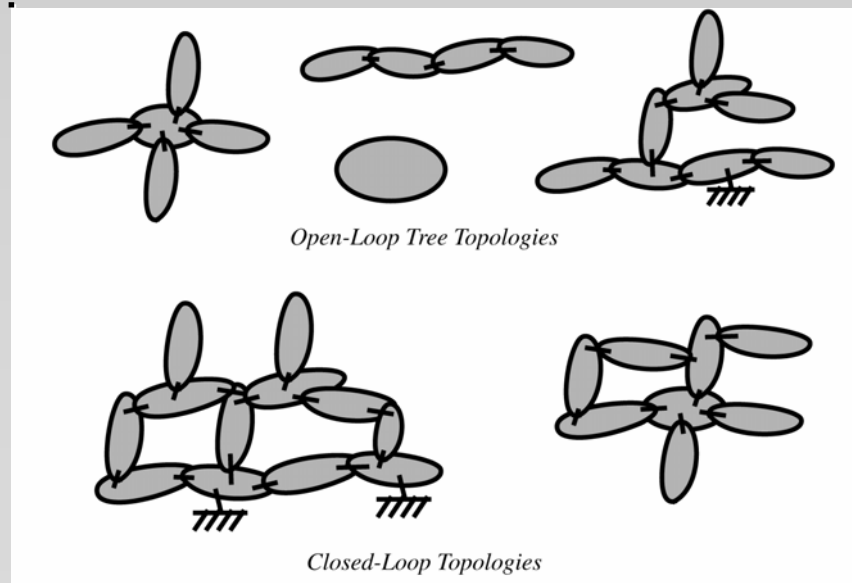
General Capabilities



SD/FAST can be used to analyze any mechanical system that can be modeled as a set of rigid bodies connected by joints, influenced by forces, driven by prescribed motions, and restricted by constraints.

This is done by specification of:

- System geometry
 - Topology
 - Joints
 - Inertial parameters
- Forces and torques
- Motion constraints
 - Joint constraints
 - Prescribed motion



Dynamic Simulation Steps

- Generate system description file (specify topology, inertial parameters, segment geometry, joints ...)
- Run SD/FAST to generate code representing the equations of motion
- Write driver program (main.c) to combine SD/FAST generated code with SD/FAST library routines and user-specific routines (e.g., muscle models, ground contact)
- Perform numerical integration of equations of motion (use default integrators or more powerful integrators)

System Description File

```
# This is an SD/FAST input file describing the pendulum
# example in Tutorial 2 of the SD/FAST User's Manual.
# The system is a one degree of freedom pendulum with
# gravity acting in the -n2 direction; the pin joint along n3.
```

```
gravity = 0 -9.8 0
```

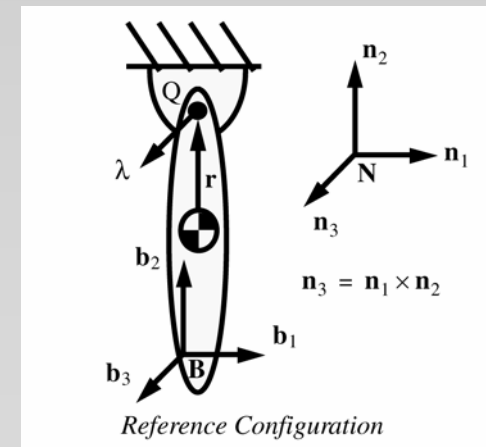
```
body = pendulum inb = $ground joint = pin
```

```
mass = 10?
```

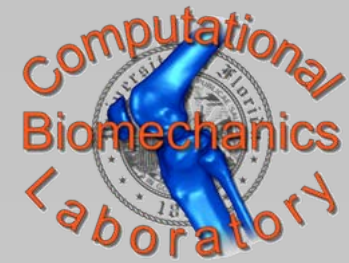
```
inertia = 5 1 5
```

```
bodytojoint = 0 1.5 0
```

```
pin = 0 0 1
```

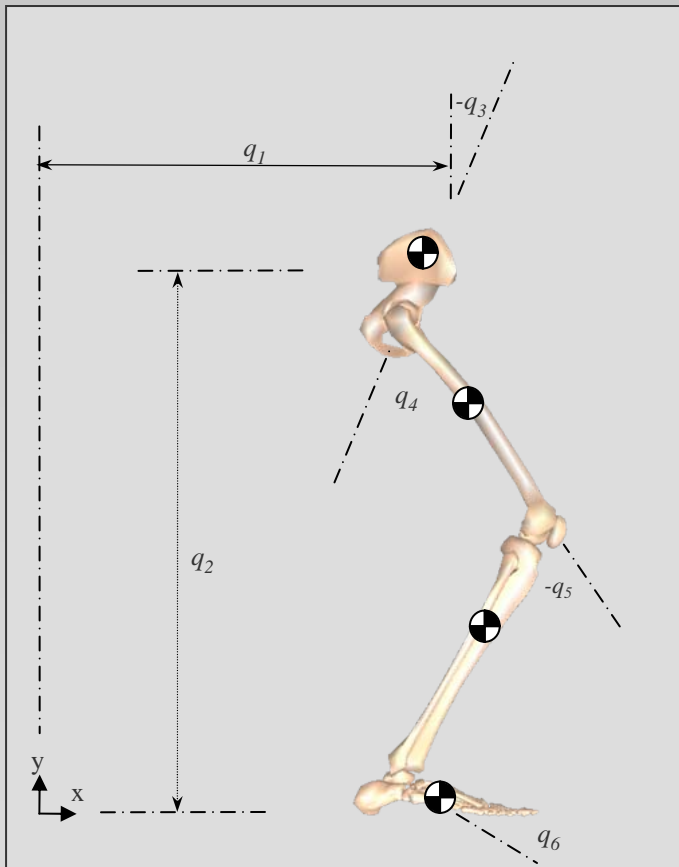


Anthropometry



- Study of the physical dimensions and inertial characteristics of the body
- Important for specifying the segment lengths and inertial parameters needed for system description file
- Values typically determined from cadaver experiments and scaled to individual subjects
- Winter (1990) is a useful reference

Swing Phase of Gait Simulation



Kinematics

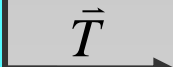
$$\bar{q}(t), \dot{\bar{q}}(t), \ddot{\bar{q}}(t)$$

Joint Torques

Inverse Dynamics

$$\bar{T} = \bar{I}(\bar{q})\ddot{\bar{q}} - \bar{C}(\bar{q}, \dot{\bar{q}}) - \bar{G}(\bar{q})$$

Forward Dynamics

$$\ddot{\bar{q}} = \bar{I}(\bar{q})^{-1} \{ \bar{C}(\bar{q}, \dot{\bar{q}}) + \bar{G}(\bar{q}) + \bar{T} \}$$


$$\bar{q}^*(t), \dot{\bar{q}}^*(t)$$



Initial Conditions

$$\bar{q}(0), \dot{\bar{q}}(0)$$

SD/Fast

main.c

```
/*  
MAIN Routine  
*/  
main(void)  
{  
    // Initialize variables  
    int i, j, ny, num_frames, err, which;
```

-
-

```
// Read in kinetics data  
load_kinetics_data(kinetics_file,nq,num_frames);
```

```
// Initialize the SD/FAST model  
sdinit();
```

-
-

gait_swing.data

datarows 41

datacolumns 19

otherdata 7

frame	pelvis_tx	pelvis_ty	pelvis_rotation	hip_angle	
60	0.890	0.894	5.000	-12.091	...
61	0.906	0.895	5.000	-10.384	...
62	0.923	0.896	5.000	-8.554	...
63	0.939	0.897	5.000	-6.623	...
...					...
...					...

SDINIT

Call before beginning a simulation:

- Computes fixed quantities (e.g. system mass)
- Normalizes pin vectors
- Verifies that ? parameters from system description file have been given values

main.c – Inverse Dynamics

```
// Step through all the data frames
for (i=0; i<=num_frames; i++) {

    // Get the experimental state and accelerations
    get_data_frame(y,dy,i);

    // Set the SD/FAST system state
    sdstate(time[i],y,&y[nq]);

    // Calculate joint torques
    sdcomptrq(&dy[nq],torque);

    // Save the torque data for use in forward dynamics
    for (j=0;j<nq;j++)
        torque_data[j][i]=torque[j];

    // Output data to motion frame
    write_motion_frame(fp_inv,time[i],y,dy,torque);

}
```

GET_DATA_FRAME

$$\vec{y} = \begin{pmatrix} \vec{q}(t) \\ \dot{\vec{q}}(t) \end{pmatrix} \quad \dot{\vec{y}} = \begin{pmatrix} \dot{\vec{q}}(t) \\ \ddot{\vec{q}}(t) \end{pmatrix}$$

SDSTATE()

- Set the system state which is saved for later use
- Applied forces, torques set to zero
- Prescribed accelerations set to zero

SDCOMPTRQ()

- Solves the joint torques required to produce the specified accelerations

$$\vec{T} = \vec{I}(\vec{q})\ddot{\vec{q}} - \vec{C}(\vec{q},\dot{\vec{q}}) - \vec{G}(\vec{q})$$

main.c – Forward Dynamics

```
// Initial state
```

```
get_data_frame(y,dy,0);
```

```
// Step through all the data frames
```

```
for (i=0; i<num_frames; i++) {
```

```
    // Current frame and time
```

```
    current_frame = i;
```

```
    t=i*T;
```

```
    // Calculate current state derivatives
```

```
    calc_deriv(t,y,dy,param,&err);
```

```
    // Numerically integrate across the current time step using a variable step integrator
```

```
    sdvinteg(calc_deriv, &t, y, dy, param, T, &step, ny, tol, work, &err, &which);
```

```
    // Output data to motion frame
```

```
    write_motion_frame(fp_for,time[i],y,dy,torque);
```

```
}
```

GET_DATA_FRAME

Initial state from experimental values

$$\vec{y} = \begin{pmatrix} \bar{q}(0) \\ \dot{\bar{q}}(0) \end{pmatrix}$$

CALC_DERIV

Compute state derivatives given the current state and time

$$\dot{\vec{y}} = \vec{f}(\vec{y}, t)$$

SDVINTEG

4th order Runge-Kutta-Merson variable time step numerical integrator

$$\vec{y}(t + dt) = \vec{y}(t) + \int_t^{t+dt} \vec{f}(\vec{y}, t)$$

calc_deriv.c

```
void calc_deriv(double time, double y[], double dy[],  
               double param[], int* status)  
{
```

```
/* Specify current state for SD/FAST routines */  
sdstate(time,y,&y[nq]);
```

```
/* Set any prescribe acceleration of joints */  
sdumotion(time,y,&y[nq]);
```

```
/* Apply any forces, torques acting on system */  
sduforce(time,y,&y[nq]);
```

```
/* Calculate derivatives of body-segment states */  
sdderiv(dy,&dy[nq]);
```

```
*status = 0;
```

SDSTATE

- Set the current system state
- Applied forces, torques set to zero:
- Prescribed motions (accelerations) set to zero

SDUMOTION

```
int sdumotion(double t, double q[], double u[])  
{  
    // No prescribed accelerations in this model  
    return 1;  
}
```

```
}
```

calc_deriv.c

```
void calc_deriv(double time, double y[], double dy[],
               double param[], int* status)
{

/* Specify current state for SD/FAST routines */
sdstate(time,y,&y[nq]);

/* Set any prescribe acceleration of joints */
sdumotion(time,y,&y[nq]);

/* Apply any forces, torques acting on system */
sduforce(time,y,&y[nq]);

/* Calculate derivatives of body-segment states */
sdderiv(dy,&dy[nq]);

*status = 0;

}
```

SDUFORCE

```
int sduforce(double t, double q[], double u[]) {
    int j;
    double torque;
    // Apply joint torques
    for (j=0;j<nq;j++) {
        torque = torque_data[j][current_frame];
        sdhinger(joint[j],axis[j],torque);
    }
    return 1;
}
```

SDHINGET

- Applies a ‘torque’ at a particular hinge of a joint
- Other commands (*sdpointf*, *sdbodyt*) exist for applying point forces and pure body torques.

calc_deriv.c

```
void calc_deriv(double time, double y[], double dy[],  
               double param[], int* status)
```

```
{
```

```
/* Specify current state for SD/FAST routines */
```

```
sdstate(time,y,&y[nq]);
```

```
/* Set any prescribe acceleration of joints */
```

```
sdumotion(time,y,&y[nq]);
```

```
/* Apply any forces, torques acting on system */
```

```
sduforce(time,y,&y[nq]);
```

```
/* Calculate derivatives of body-segment states */
```

```
sdderiv(dy,&dy[nq]);
```

```
*status = 0;
```

```
}
```

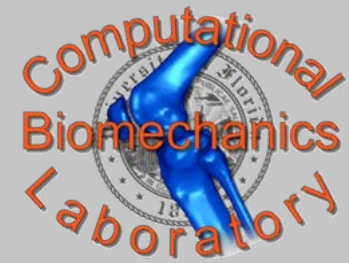
SDDERIV

Computes state derivatives (and constraint multipliers) using

- The state last passed
- Any prescribed accelerations
- Any applied forces, torques

$$\ddot{\vec{q}} = \vec{I}(\vec{q})^{-1} \{ \vec{C}(\vec{q}, \dot{\vec{q}}) + \vec{G}(\vec{q}) + \vec{T} \}$$

For Next Time



- Download SD/Fast user guide and read through Tutorials 1 and 2
- Download and read Kuo (1998) and Cahouet *et al.* (2002) articles on inverse dynamics