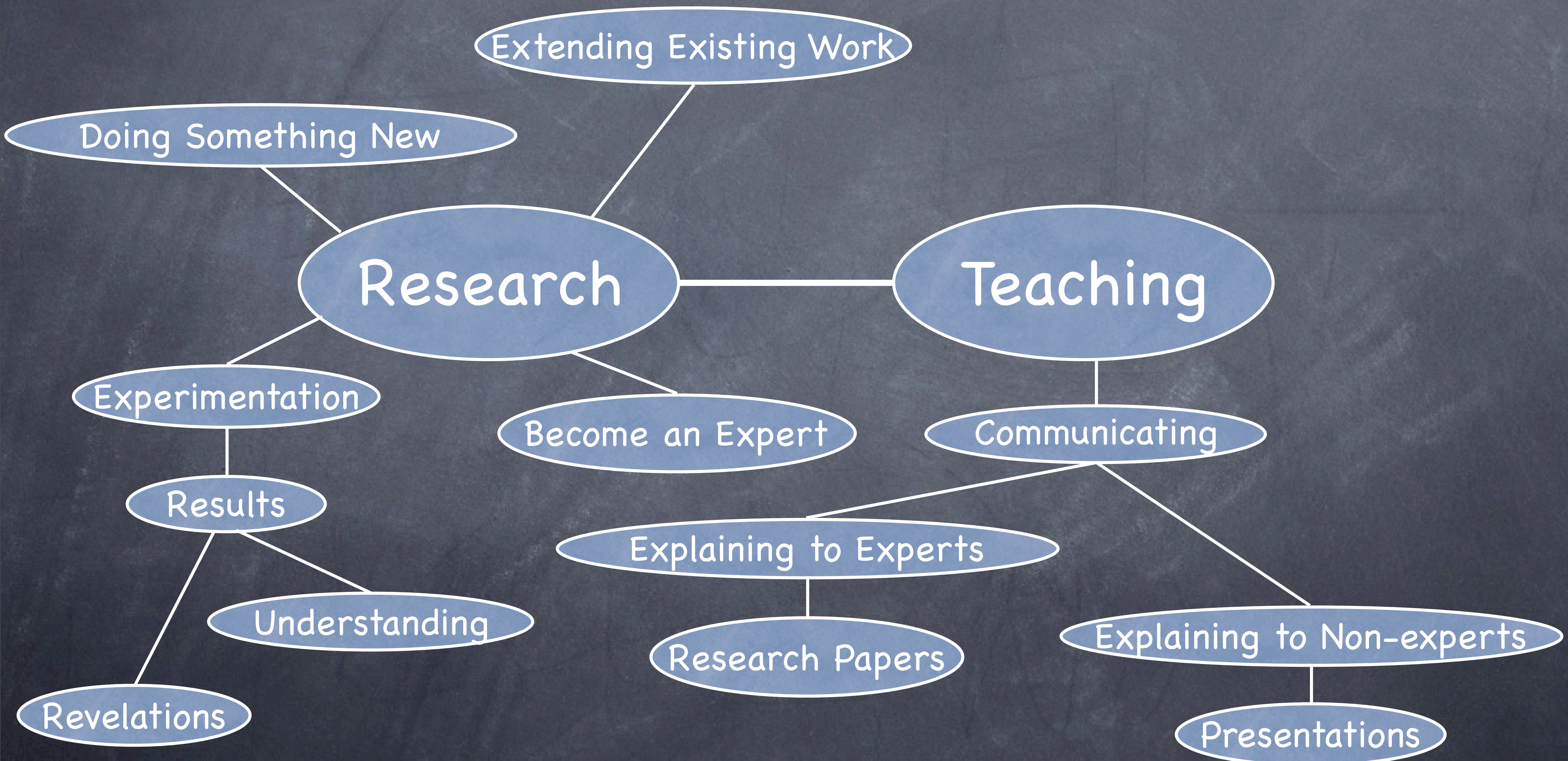
Reasoning About Programs

Mark Wheelhouse

PhD Student
Department of Computing
Imperial College London

So What is a PhD Anyway?



What am I Doing?

"Context Logic, Tree Update and Concurrency"

What am I Doing?

"Context Logic, Tree Update and Concurrency"

Huh?!

What am I Doing?

"Context Logic, Tree Update and Concurrency"

Huh?!

What's He Talking About?!

What am I Doing?

"Context Logic, Tree Update and Concurrency"

Huh?!

What's He Talking About?!

Using Mathematics to prove properties about computer programs

Sometimes we want to be 100% sure the computer is doing what it should be

Why Prove Programs?
A History Lesson

Why Prove Programs? A History Lesson

1980's - Therac 25

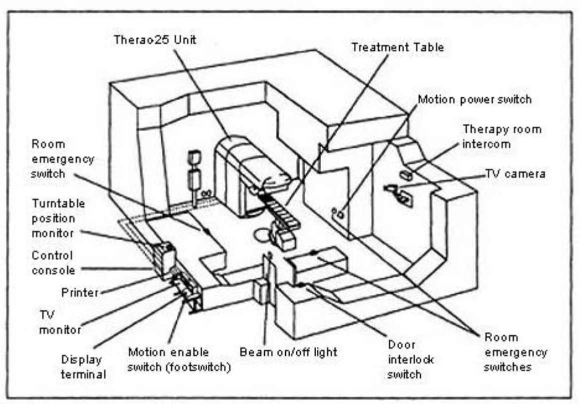


Figure 1. Typical Therac-25 facility

Why Prove Programs? A History Lesson

1980's - Therac 25

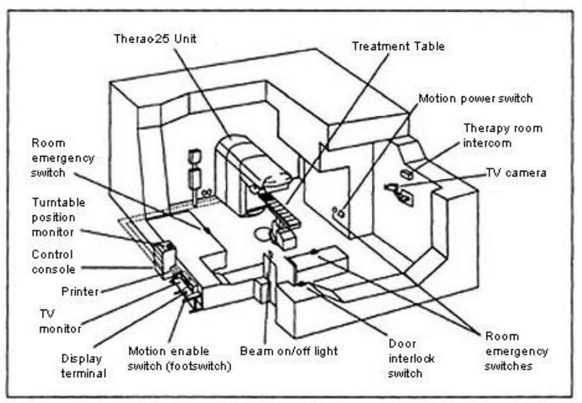


Figure 1. Typical Therac-25 facility

race condition

Why Prove Programs?
A History Lesson

1980's - Therac 25

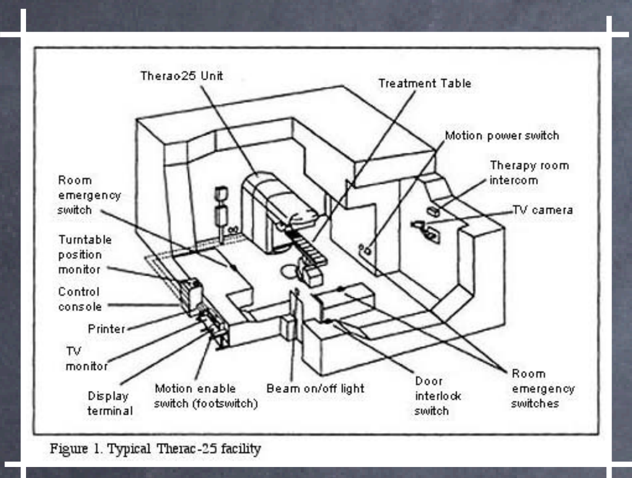
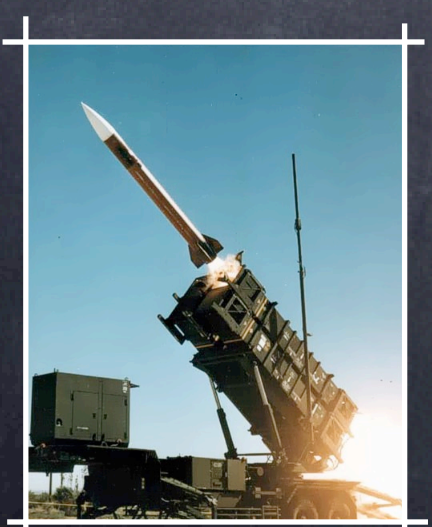


Figure 1. Typical Therac-25 facility

race condition

1991 - MIM-104 Patriot



Why Prove Programs? A History Lesson

1980's - Therac 25

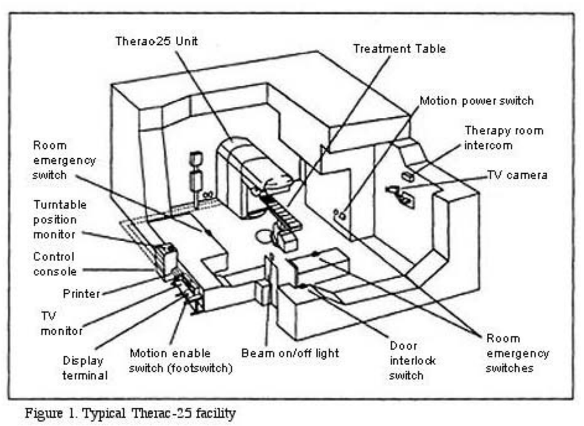
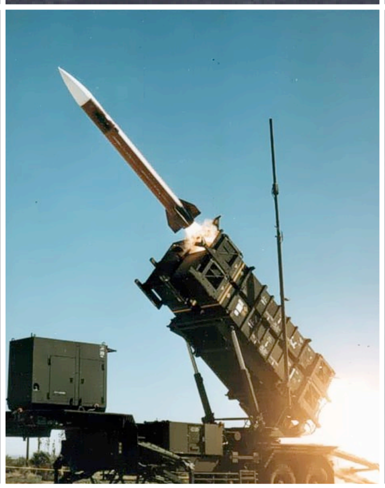


Figure 1. Typical Therac-25 facility

race condition

1991 - MIM-104 Patriot



rounding error

Why Prove Programs? A History Lesson

1980's - Therac 25

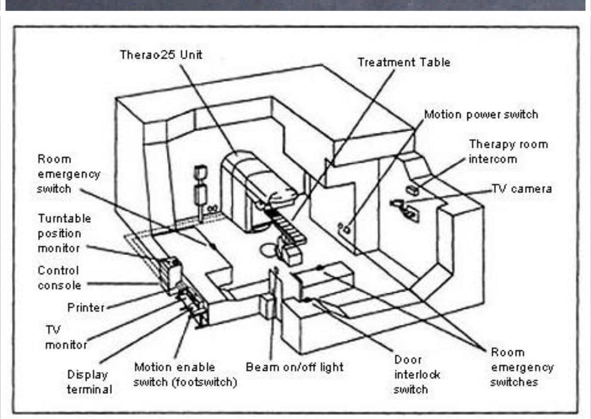


Figure 1. Typical Therac-25 facility

race condition

1996 - Ariane 5

1991 - MIM-104 Patriot

rounding error

Why Prove Programs?
A History Lesson

1980's - Therac 25

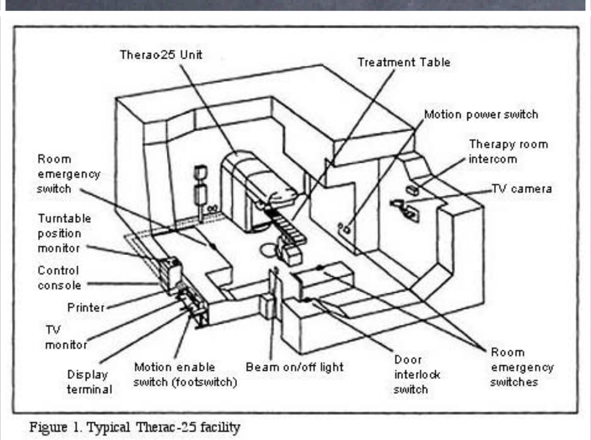


Figure 1. Typical Therac-25 facility

race condition

1996 - Ariane 5

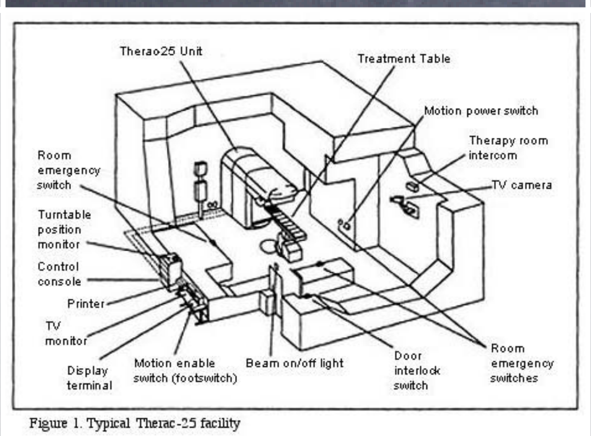
data overflow!

1991 - MIM-104 Patriot

rounding error

Why Prove Programs?
A History Lesson

1980's - Therac 25



race condition

1996 - Ariane 5

data overflow!

1991 - MIM-104 Patriot

rounding error

1997 - USS Yorktown

Why Prove Programs?
A History Lesson

1980's - Therac 25

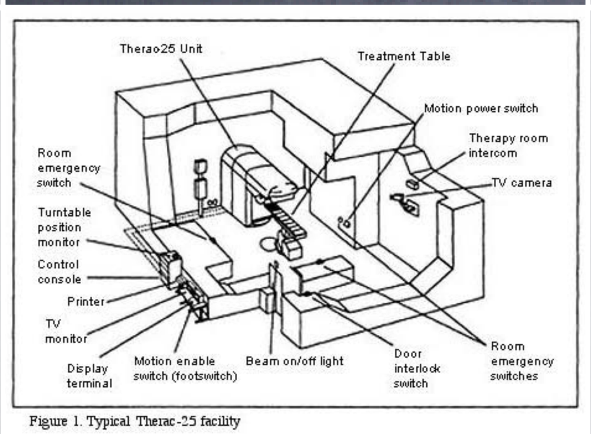


Figure 1. Typical Therac-25 facility

race condition

1996 - Ariane 5

data overflow!

1991 - MIM-104 Patriot

rounding error

1997 - USS Yorktown

divide by zero error!

What Kind of Systems do We Want to be Sure of?

What Kind of Systems do We Want to be Sure of?

Internet Banking

Automatic Breaking

Nuclear Power Stations

Flight Control Systems

Building Design Software

IFF Targeting Systems

Heart Monitors

Spaceships

Pacemakers

What Kind of Systems do We Want to be Sure of?

Internet Banking

Automatic Breaking

Nuclear Power Stations

Flight Control Systems

Building Design Software

IFF Targeting Systems

Heart Monitors

Spaceships

Pacemakers

i.e. - Anything that is safety critical

- When the cost of system failure is huge

Spotting Failure

- We want to spot when a system can fail.
- We want to prove that a system will not fail.
- We can use mathematics (namely logic) to do this.

Logic - The Basics

boolean variables - “have one of two values”

P =	**true**	**false**
	1	0

Logic - The Basics

boolean variables - "have one of two values"

P =	**true**	**false**
	1	0

boolean operators - "ways of combining variables"

and: $\wedge$ or: $\vee$ implies: $\Rightarrow$

Logic - The Basics

boolean variables - "have one of two values"

	true	**false**
p =	1	0

boolean operators - "ways of combining variables"

and: $\wedge$

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

or: $\vee$

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

implies: $\Rightarrow$

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Logic - Predicates

X,Y,Z are things (objects, people, concepts,...)

isRed(X) = true if X is red

isRound(X) = true if X is round

isRedBall(X) = true if X is a red ball

Logic - Predicates

X,Y,Z are things (objects, people, concepts,...)

isRed(X) = true if X is red

isRound(X) = true if X is round

isRedBall(X) = true if X is a red ball

$$isRedBall(X) \Rightarrow isRed(X) \wedge isRound(X)$$

Logic - Structures

another way of looking at this using a satisfaction relation,

$$\models$$

$X \models \text{isRed} \quad \Leftrightarrow \quad X \text{ is red}$

$X \models \text{isRound} \quad \Leftrightarrow \quad X \text{ is round}$

$X \models \text{isRedBall} \quad \Leftrightarrow \quad X \text{ is a red ball}$

Logic - Structures

another way of looking at this using a satisfaction relation,

$$\models$$

$X \models \text{isRed} \quad \Leftrightarrow \quad$ X is red

$X \models \text{isRound} \quad \Leftrightarrow \quad$ X is round

$X \models \text{isRedBall} \quad \Leftrightarrow \quad$ X is a red ball

$$\text{isRedBall} \Rightarrow \text{isRed} \wedge \text{isRound}$$

Logic - Structures

data $\models$ Tree-Formula

tree T ::=

Logic - Structures

data $\models$ Tree-Formula

tree T ::= 0 empty tree

Logic - Structures

data $\models$ Tree-Formula

tree	T ::=	0	empty tree
		n[T]	tree node

Logic - Structures

data $\models$ Tree-Formula

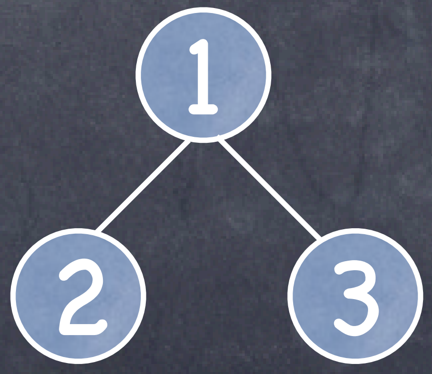
tree	T ::=	0	empty tree
		n[T]	tree node
		T | T	ordered trees

Logic - Structures

data $\models$ Tree-Formula

tree	T ::=	0	empty tree
		n[T]	tree node
		T | T	ordered trees

Eg: 1[2[0] | 3[0]]

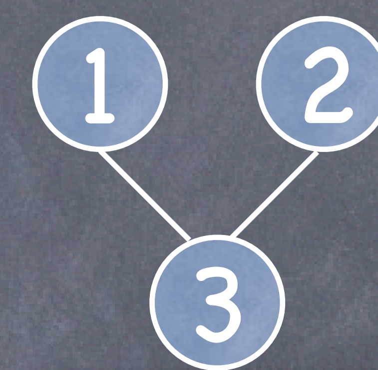


Tree Examples

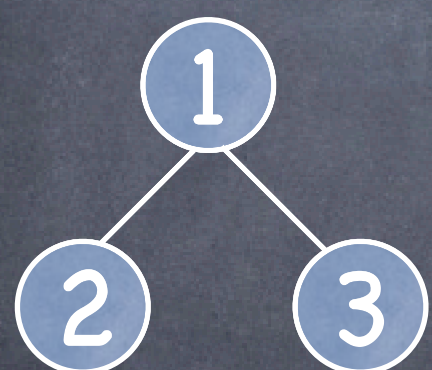
a)

b)

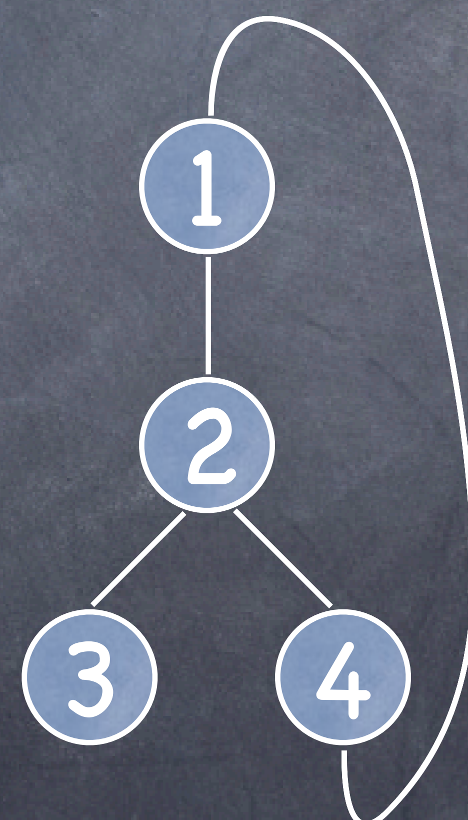
c)



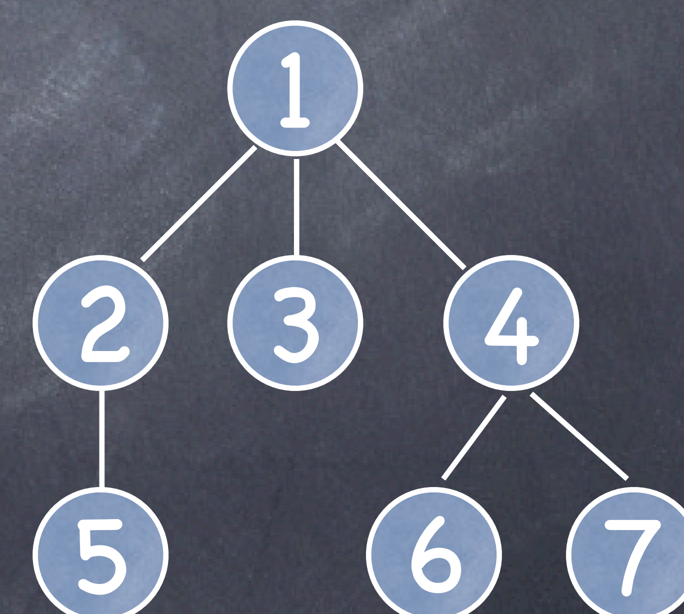
d)



e)



f)



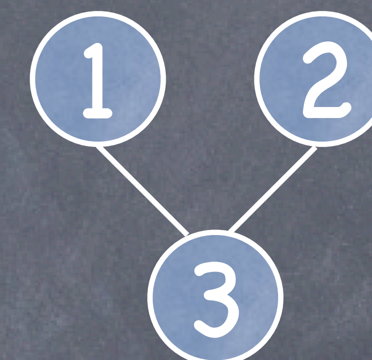
Tree Examples

a)

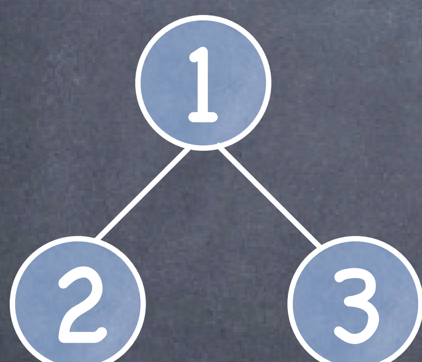
1[0]

b)

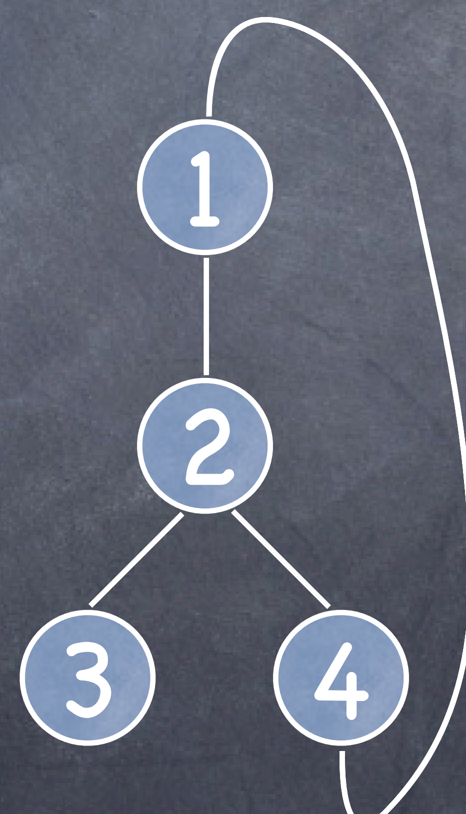
c)



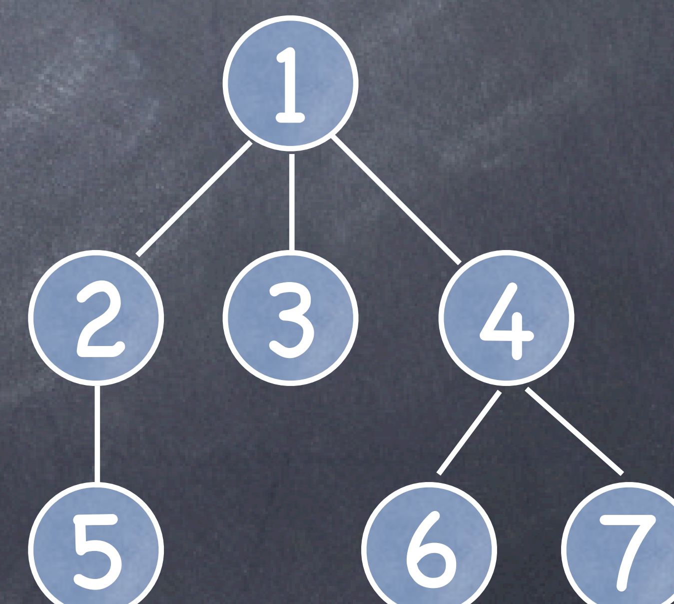
d)



e)



f)



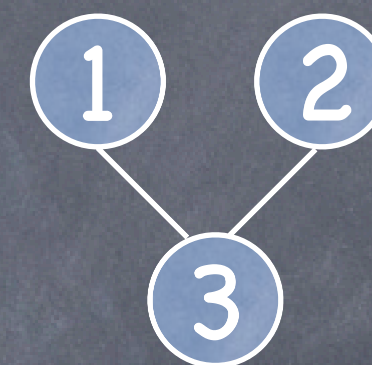
Tree Examples

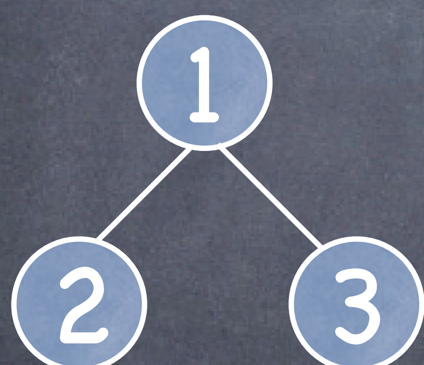
a) 

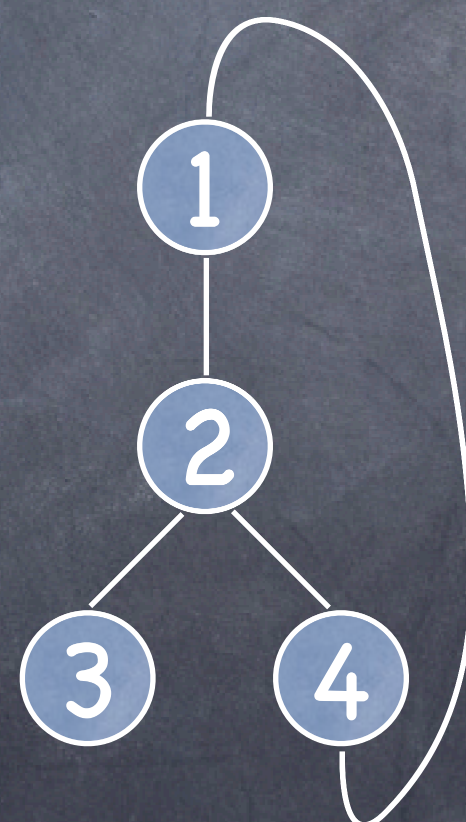
1[0]

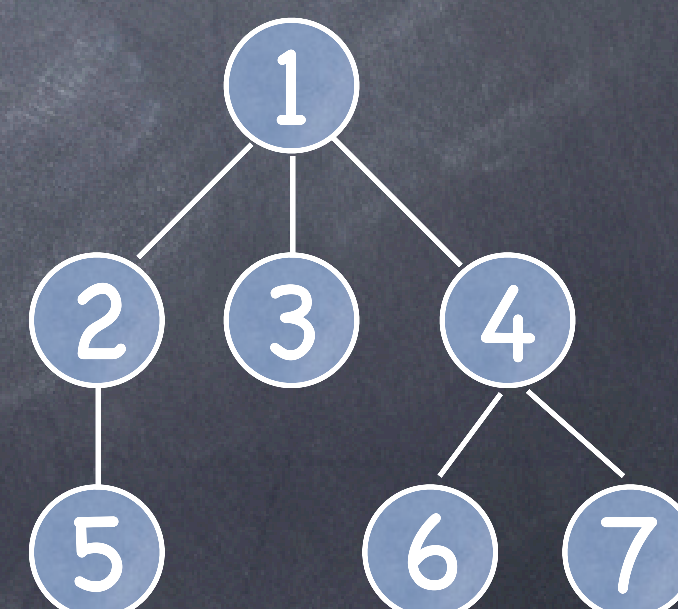
b) 

1[0] | 2[0]

c) 

d) 

e) 

f) 

Tree Examples

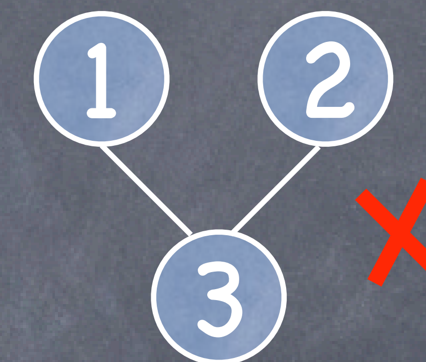
a)

1[0]

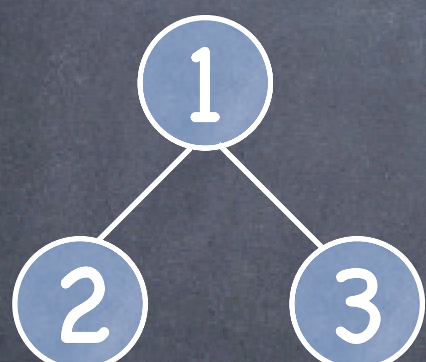
b)

1[0] | 2[0]

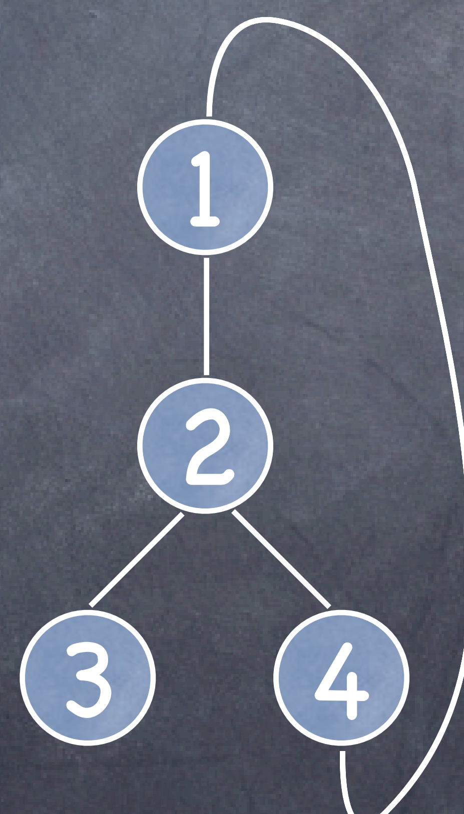
c)



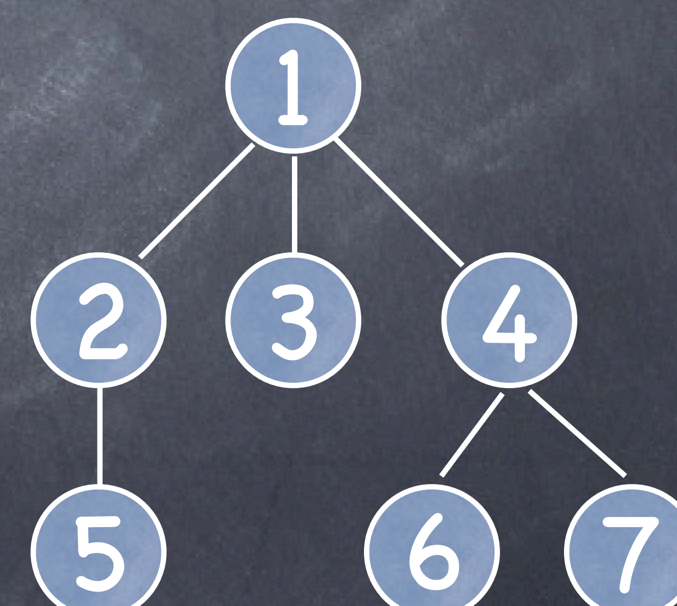
d)



e)



f)



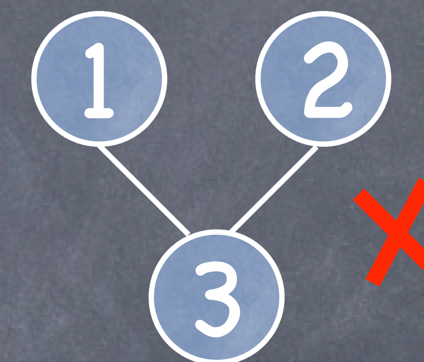
Tree Examples

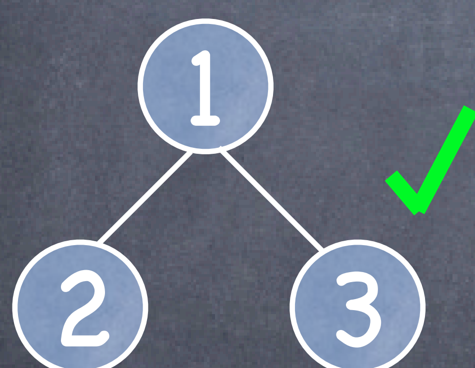
a) 

1[0]

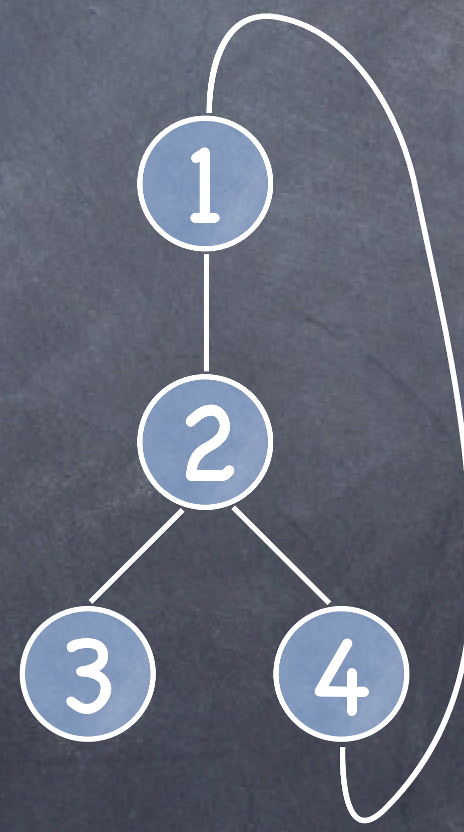
b) 

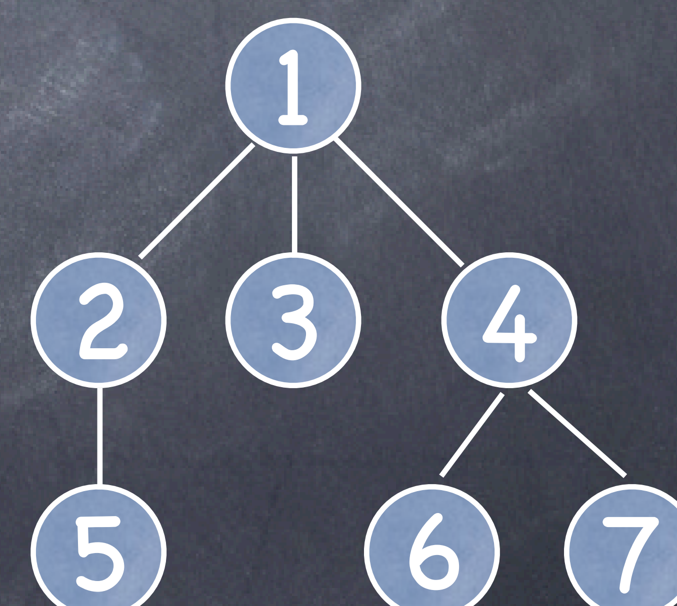
1[0] | 2[0]

c) 

d) 

1[2[0] | 3[0]]

e) 

f) 

Tree Examples

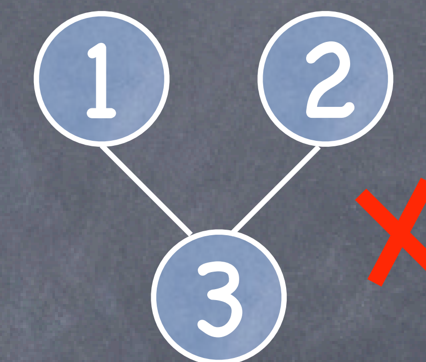
a)

1[0]

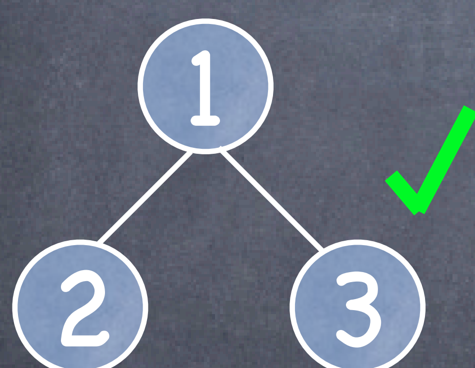
b)

1[0] | 2[0]

c)

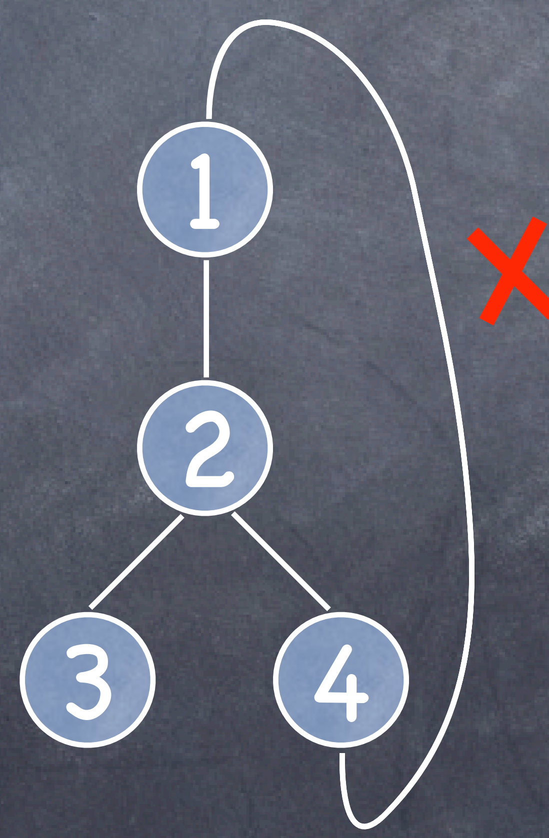


d)

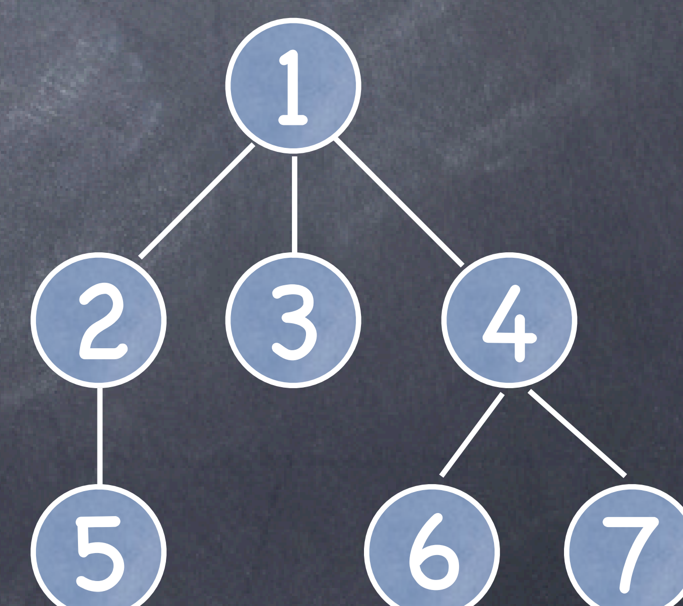


1[2[0] | 3[0]]

e)



f)



Tree Examples

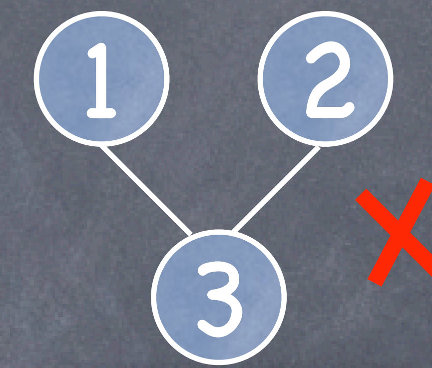
a)

1[0]

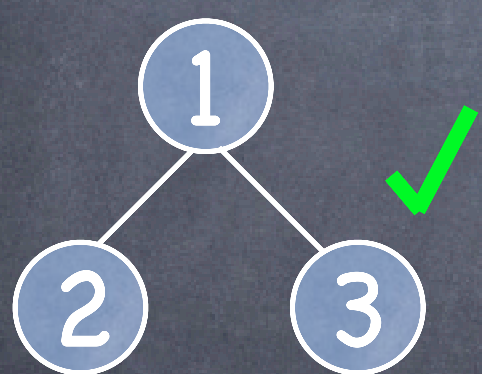
b)

1[0] | 2[0]

c)

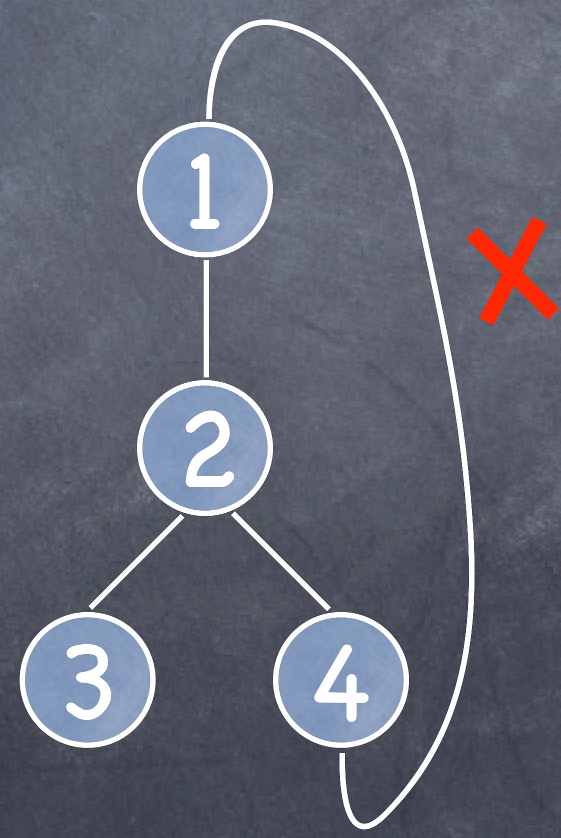


d)

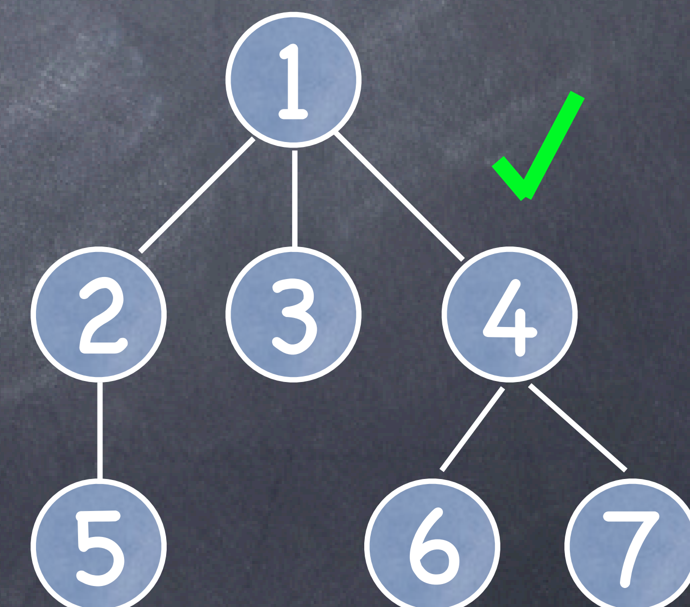


1[2[0] | 3[0]]

e)



f)



1[2[5[0]] | 3[0] | 4[6[0] | 7[0]]]

Data Update

We want to describe the behavior of a program,

Eg:

{ colour(X, ?) }

paint(X, Green)

{ colour(X, Green) }

Data Update

We want to describe the behavior of a program,

{ P } computer command { Q }

pre-condition post-condition

Eg:

{ colour(X, ?) }

paint(X, Green)

{ colour(X, Green) }

Data Update

We want to describe the behavior of a program,

{ P } computer command { Q }

pre-condition post-condition

pre-condition must hold before and post-condition must hold after, or we get a fault!

Eg:

{ colour(X, ?) }

paint(X, Green)

{ colour(X, Green) }

Local Data Update

$\{ y=3, x=n, z=2 \}$

addOne(x)

$\{ y=3, x=n+1, z=2 \}$

Local Data Update

{ y=3, x=n, z=2 }
addOne(x)
{ y=3, x=n+1, z=2 }

↓

{ x=n }
addOne(x)
{ x=n+1 }

Local Tree Update

$\{\ m[\ t_1 \mid n[t_2] \mid t_3\ ]\ \}$

p := goUpTree(n)

$\{\ m[\ t_1 \mid n[t_2] \mid t_3\ ] \wedge (p=m)\ \}$

$\{\ n[t]\ \}$

deleteTree(n)

$\{\ 0\ \}$

$\{\ n[t]\ \}$

addNodeAfter(n,x)

$\{\ n[t] \mid x[0]\ \}$

Example - Family Trees

A more complex data structure,

Example - Family Trees

A more complex data structure,

database D ::=

Example - Family Trees

A more complex data structure,

database D ::= 0_D empty database

Example - Family Trees

A more complex data structure,

database	D ::=	0_D	empty database
		familyName[F]	one family

Example - Family Trees

A more complex data structure,

database	$D ::=$	0_D	empty database
		familyName[F]	one family
		$D + D$	database join

Example - Family Trees

A more complex data structure,

database	$D ::=$	0_D	empty database
		familyName[F]	one family
		$D + D$	database join

family tree $F ::=$

Example - Family Trees

A more complex data structure,

database $D ::=$	0_D	empty database
	familyName[F]	one family
	$D + D$	database join
family tree $F ::=$	0_F	empty tree

Example - Family Trees

A more complex data structure,

database D ::=	0_D	empty database
	familyName[F]	one family
	D + D	database join
family tree F ::=	0_F	empty tree
	name:marTo Gen:Age [F]	tree node

Example - Family Trees

A more complex data structure,

database D ::=	0_D	empty database
	familyName[F]	one family
	D + D	database join
family tree F ::=	0_F	empty tree
	name:marTo Gen:Age [F]	tree node
	F | F	ordered trees

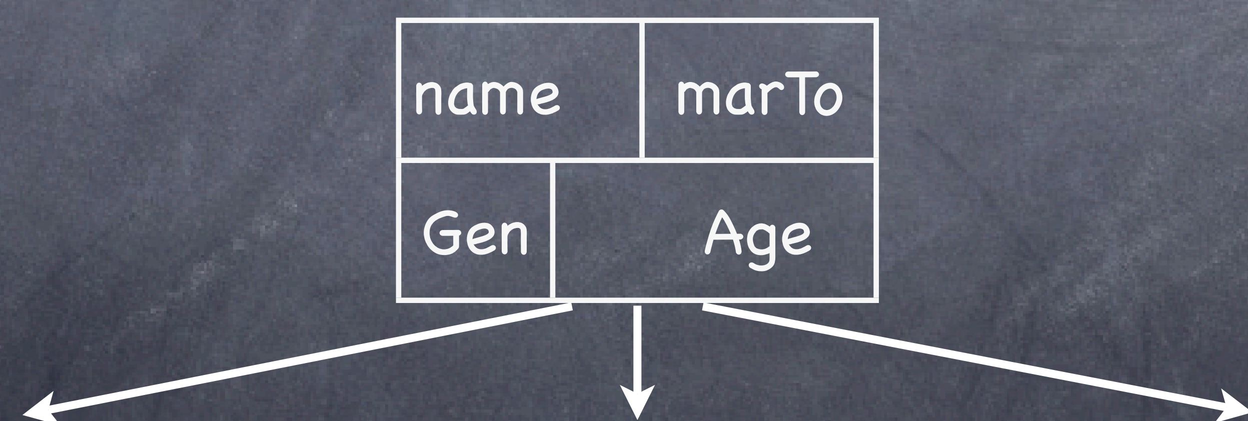
Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

II

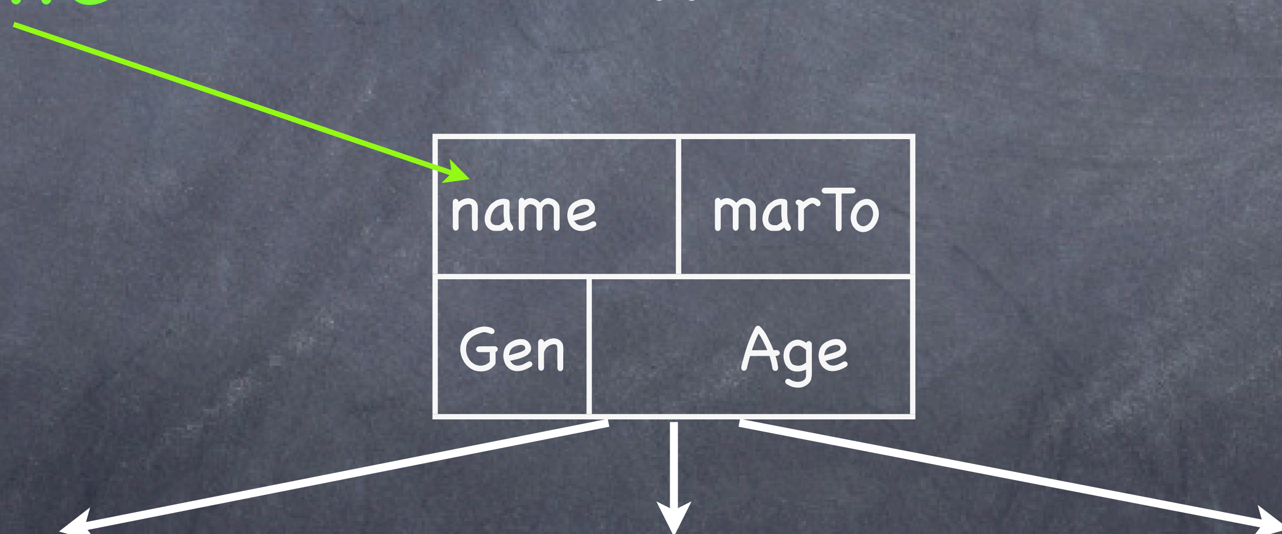


Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

||

person's name



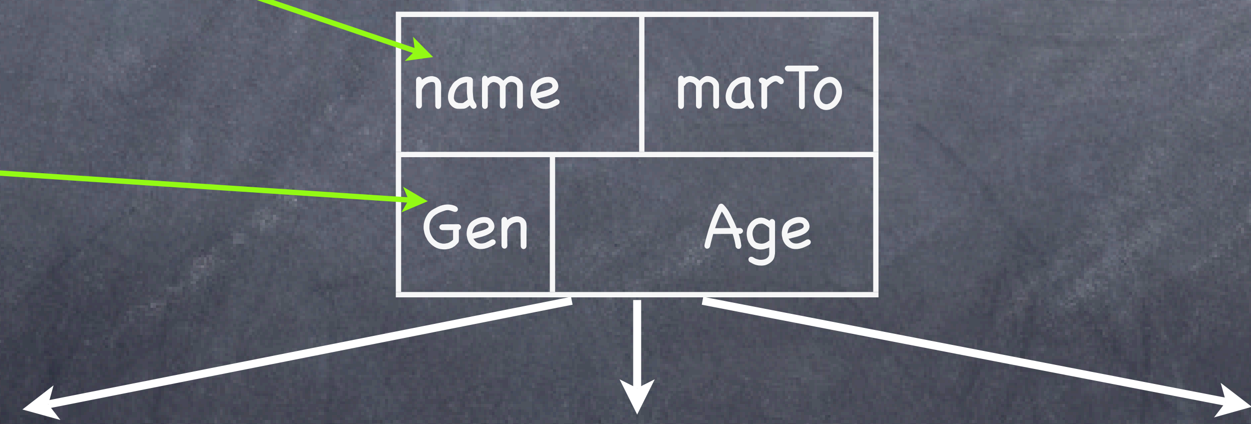
Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

=

person's name

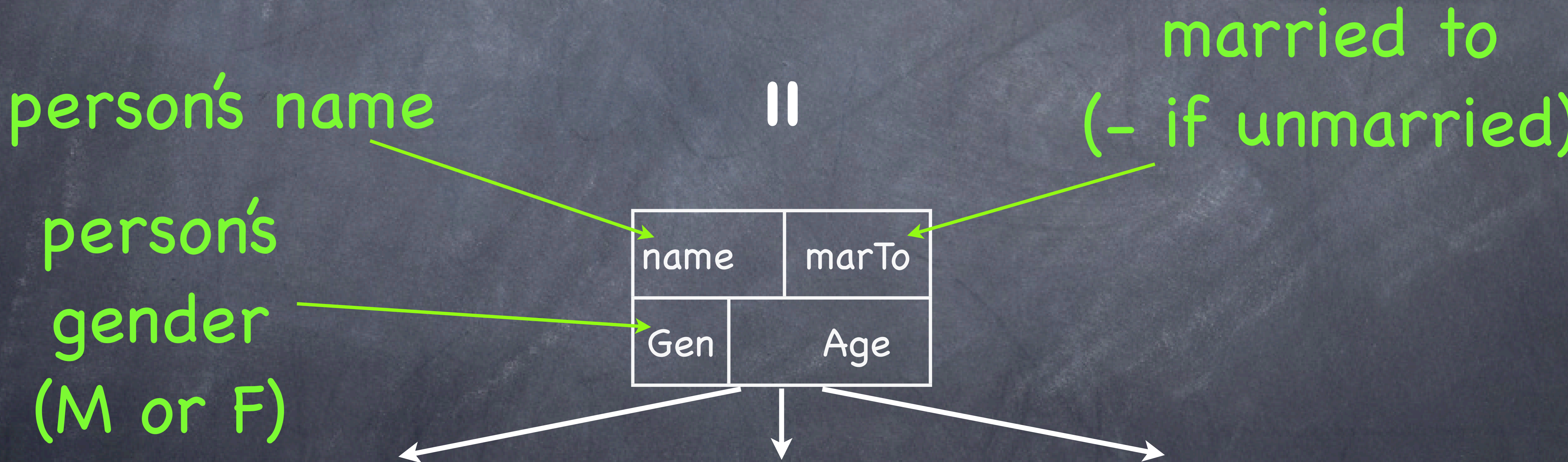
person's gender (M or F)



Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

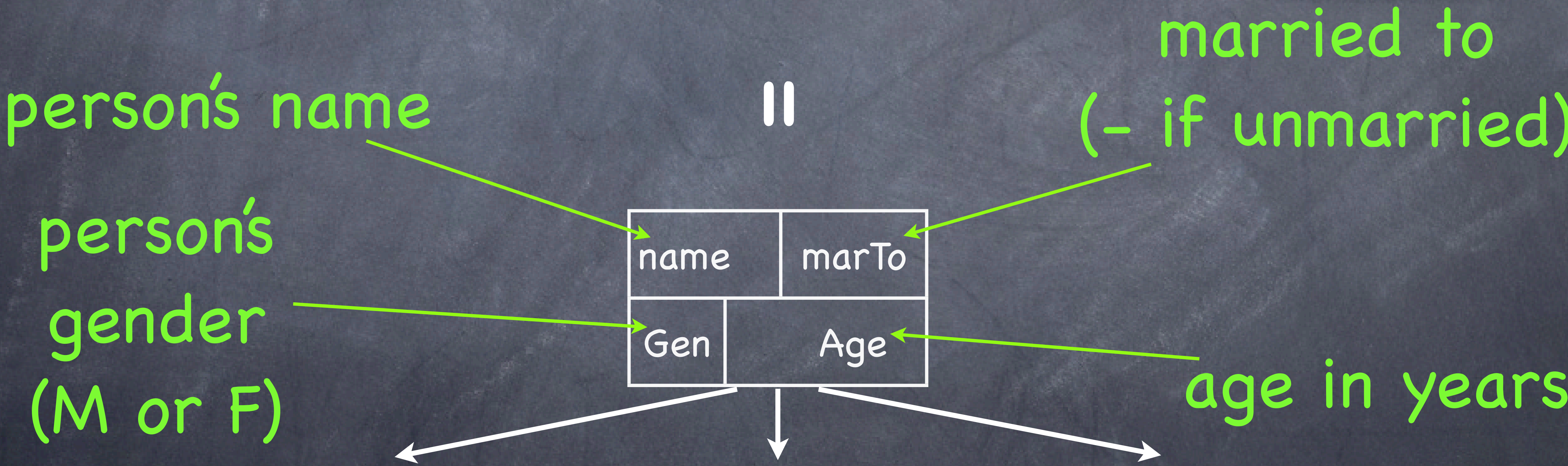
=



Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

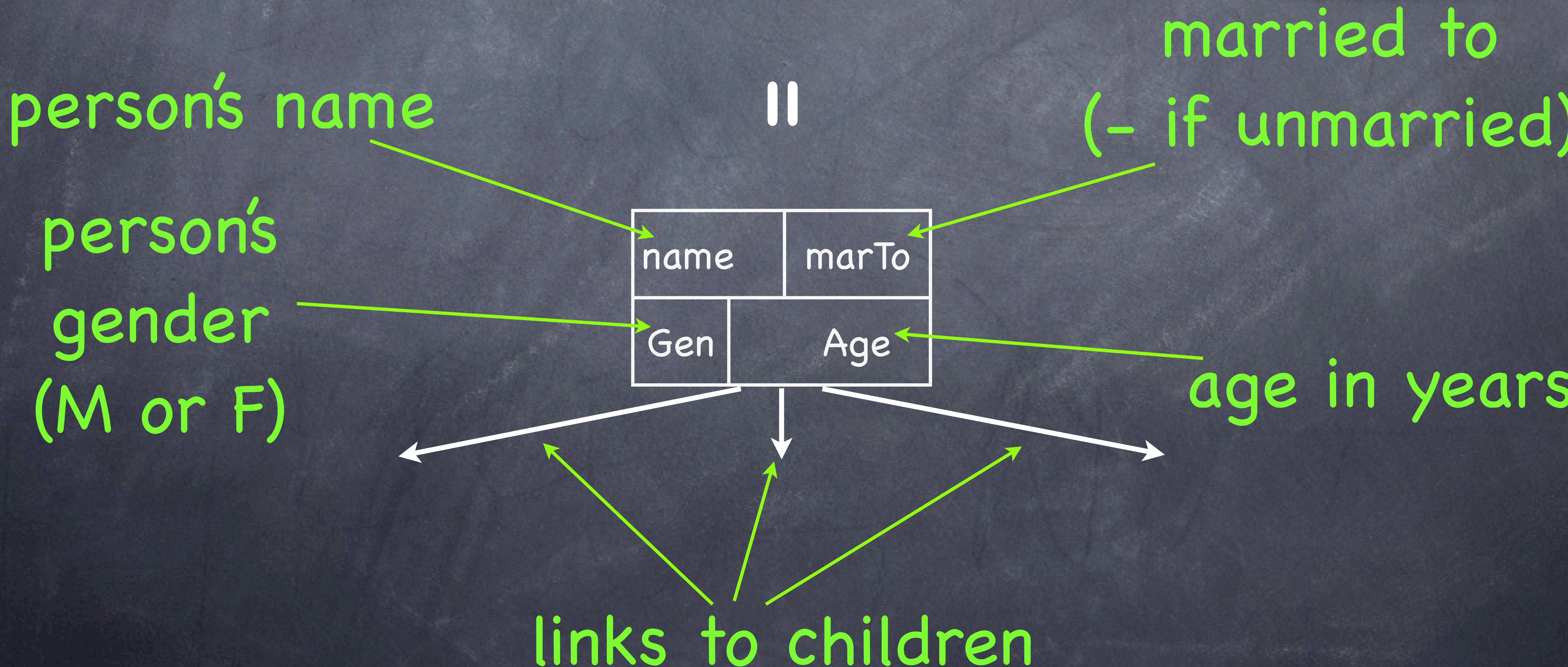
=



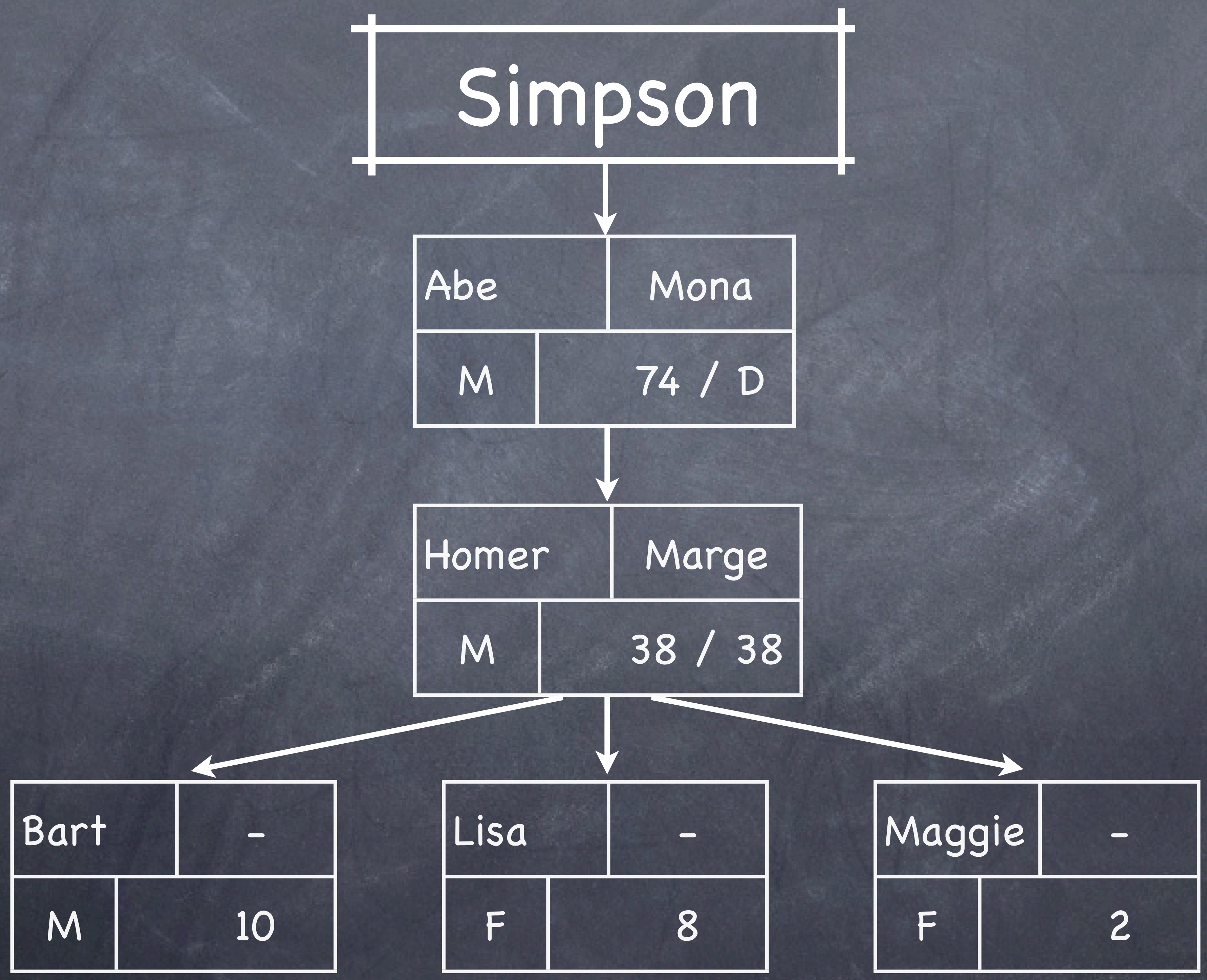
Example - Family Tree Nodes

name:marTo
Gen:Age [F] tree node

=

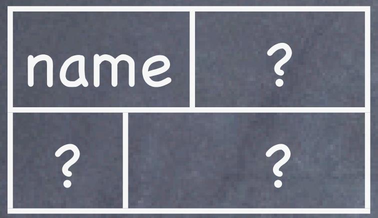


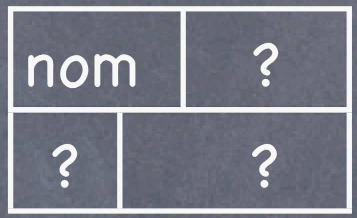
Example - The Simpsons



Example - Commands

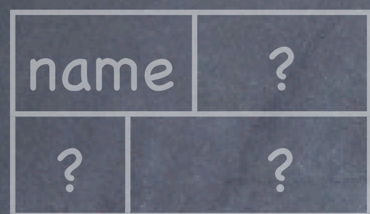
Example - Commands

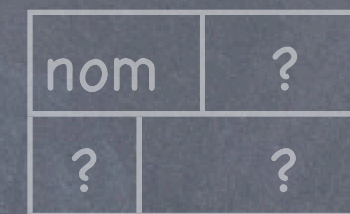
{ 

} reName(name, nom) { 

}

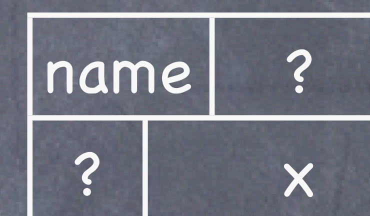
Example - Commands

{ 

} reName(name, nom) { 

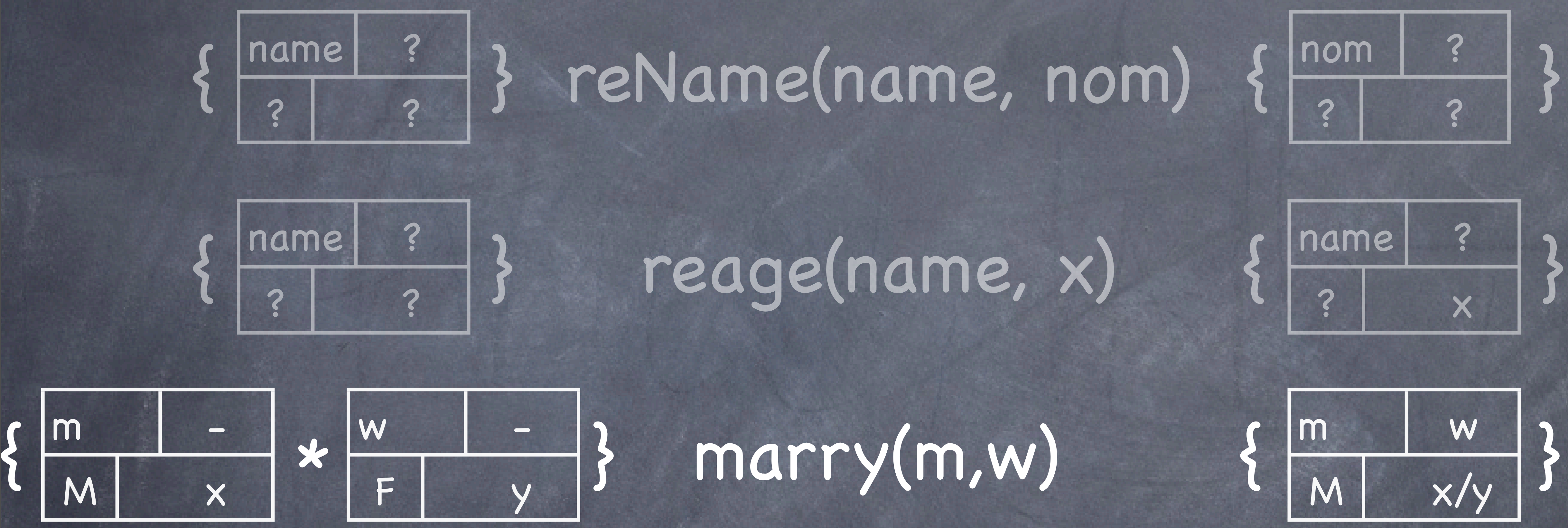
}

{ 

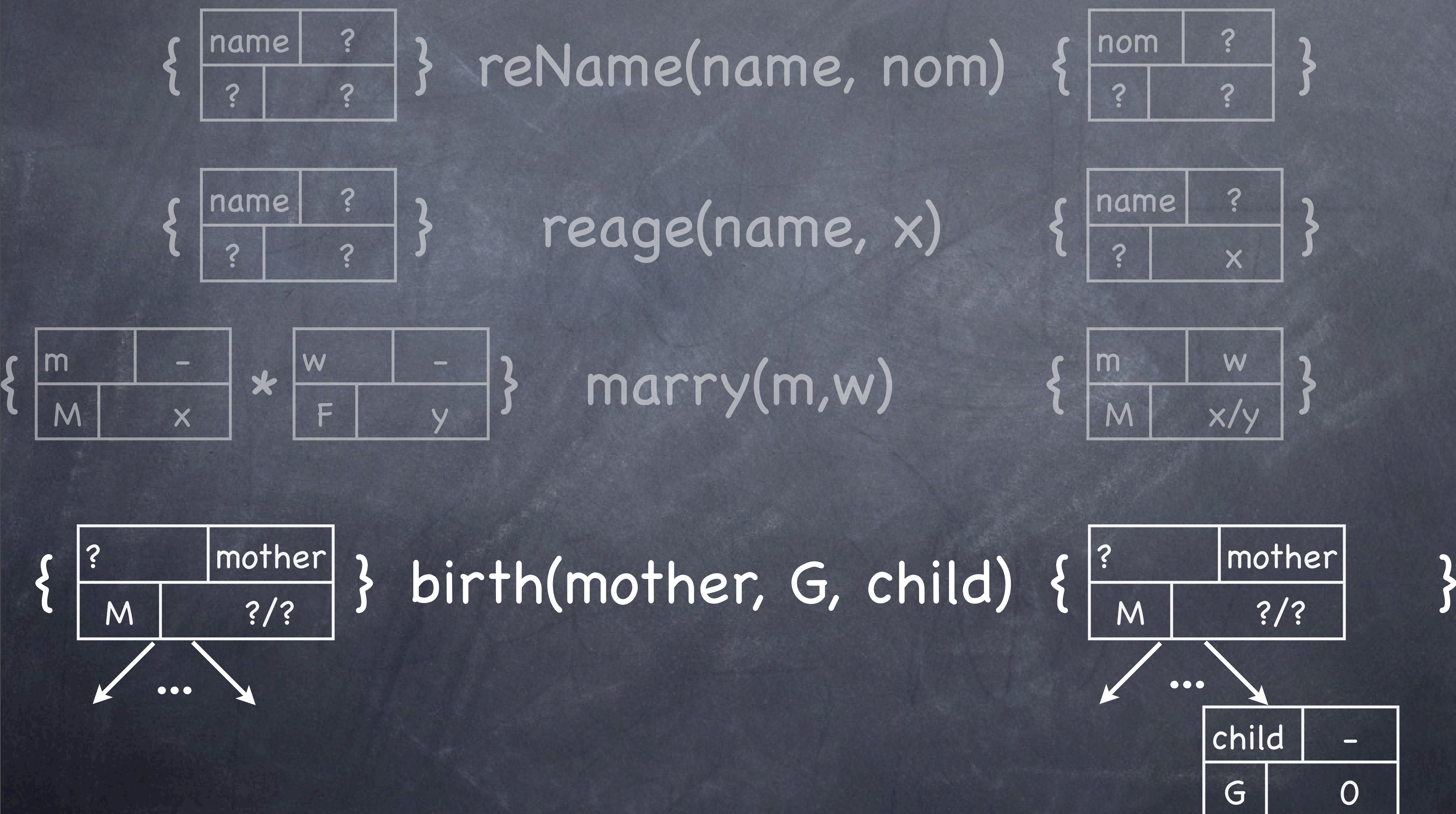
} reage(name, x) { 

}

Example - Commands



Example - Commands



Now you are going to perform concurrent local reasoning!

Example - Commands