Microsimulations in R

Abhijit Dasgupta, PhD
ARAASTAT

For Pharmerit, LLC

Random numbers in R

- R provides random number generators for a vast number of probability distributions
- There is a selection of base pseudo-random number generating algorithms available in R (see here for more details)
- These RNG are the basis for stochastic simulation in R

A case study ISPOR

- We want to simulate the number of handouts an ISPOR attendee collects over the course of a meeting
- We consider a population of 2500 attendees
- With some probability, attendees
  - attend each of the 3 days
  - attend each session on a day he attends
  - find available handouts
  - take a handout, if available

A case study
ISPOR

Day	Average prob of attending	Distribution of attendance prob
1	0.8	Beta(8,2)
2	0.8	Beta(8,2)
3	0.6	Beta(3,2)

A case study
ISPOR

Type of session	Avg probability of attending	Distribution of attendance prob
Plenary session	0.40	Beta(4,6)
Forum	0.40	Beta(4,6)
Workshop	0.70	Beta(7,3)
Research	0.70	Beta(7,3)
Issues panel	0.40	Beta(4,6)

A case study
ISPOR

Type of session	Average prob of available handout	Distribution of the prob
Plenary, Issues panel, Forum	0.95	Beta(19,1)
Workshop	0.80	Beta(8,2)
Research	0.75	Beta(6,2)

A case study
ISPOR

Probability of picking up an available handout was 0.50 on average, distributed as Beta(6,6)

Principles of R programming

- R is a matrix-oriented language
  - There are many efficient matrix operations in-built
  - There are also many functions that efficiently operate on vectors and matrices
  - Arranging data into vectors and matrices and operating on them makes for faster programs generally

What does this mean?

- You want to add the numbers 1-1000
- You can loop

```
sum.using.loops = function(n){
  Sum = 0
  for(i in 1:n) Sum = Sum + i}
```

- You can operate on vectors

```
sum.using.sum = function(n){
  x = 1:n
  Sum = sum(x)}
```

What does this mean?

- You want to add the numbers 1-1000
- You can loop

```
sum.using.loops = function(n){
  Sum = 0
  for(i in 1:n) Sum = Sum + i}
```

- You can operate on vectors

```
sum.using.sum = function(n){
  x = 1:n
  Sum = sum(x)}
```

```
                 test replications elapsed relative
  sum.using.sum(1000)          100   0.001        1
sum.using.loops(1000)          100   0.080       80
```

What does this mean?

- You want to add the numbers 1-1000
- You can loop

```
sum.using.loops = function(n){
  Sum = 0
  for(i in 1:n) Sum = Sum + i}
```

- You can operate on vectors

```
sum.using.sum = function(n){
  x = 1:n
  Sum = sum(x)}
```

```
                test replications elapsed relative
  sum.using.sum(1000)          100   0.001        1
sum.using.loops(1000)          100   0.080       80
```

What does this imply?

- You have a choice
  - Generate each person’s experience one at a time, and loop
  - Generate everyone’s experience together using vectors and vector operations

Let’s do this

What does this imply?

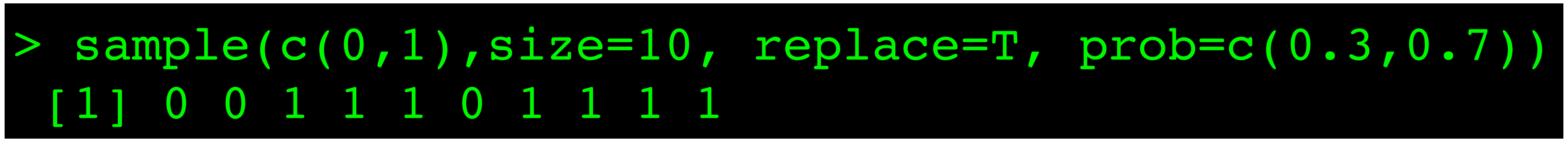
- You have a choice
  - Generate each person's experience one at a time, and loop
  - Generate everyone's experience together using vectors and vector operations

This is better

Let's do this

Generating yes/no data

- Computers don’t understand “yes” and “no”
  - This becomes 0 and 1
  - Use the function `sample`



```
> sample(c(0,1),size=10, replace=T, prob=c(0.3,0.7))
 [1] 0 0 1 1 1 0 1 1 1 1
```

  - This means
    - Sample 0’s and 1’s....
    - Generate 10 numbers....
    - with replacement....
    - with chance of a 1 being 70%

Generating yes/no data

- Turns out the function `sample.int` is about 40% faster

```
> sample.int(2,size=10,replace=T, prob=c(0.3,0.7))-1
 [1] 0 1 0 1 1 1 1 1 1 1
```

Generating yes/no data

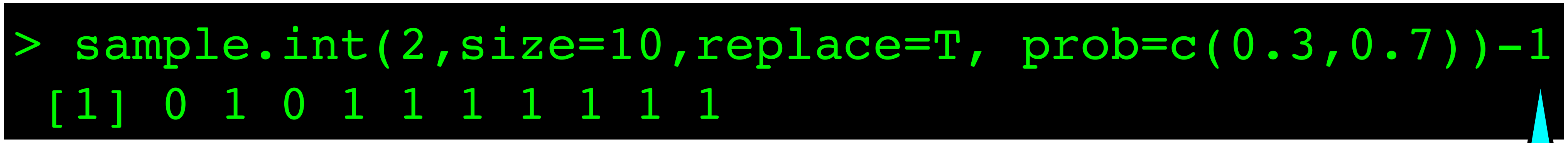
- Turns out the function `sample.int` is about 40% faster

```
> sample.int(2,size=10,replace=T, prob=c(0.3,0.7))-1
 [1] 0 1 0 1 1 1 1 1 1 1
```

This call to `sample.int` would generate 1's and 2's, not 0's and 1's. So we subtract 1

Generating yes/no data

- Turns out the function `sample.int` is about 40% faster



```
> sample.int(2,size=10,replace=T, prob=c(0.3,0.7))-1
 [1] 0 1 0 1 1 1 1 1 1 1
```

This call to `sample.int` would generate 1's and 2's, not 0's and 1's. So we subtract 1

Generally, `sample.int(`*`n, size=N, replace=T`*`)` would generate *N* numbers in *1*,2,…,*n* with replacement and equal probability; the probabilities can be changed by adding the *prob* option with a vector of probabilities.

Let’s figure out Day 1

- We need a yes/no on who, among the 2500, attended Day 1

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

- So if person 1 went to Day1,

    ```
    attended.Day1[1] is 1
    ```

    otherwise

    ```
    attended.Day1[1] is 0
    ```

Let’s figure out Day 1

- We need a yes/no on who, among the 2500, attended Day 1

```
attended.Day1 = sample.int(2, size=2500,
                          replace=T,
                          prob=c(1-p.Day1,p.Day1))-1
```

- So if person 1 went to Day1,

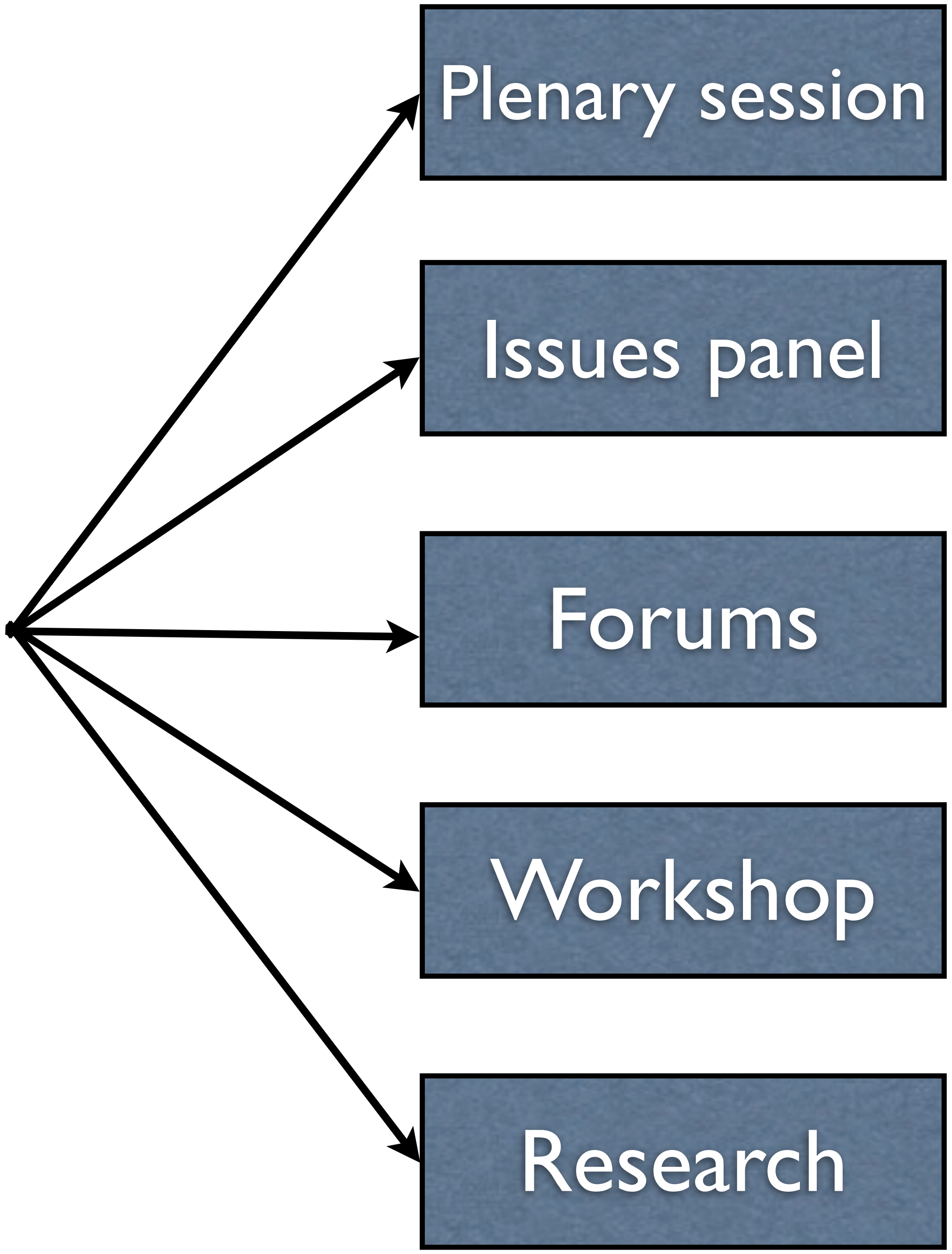
  `attended.Day1[1] is 1`

  otherwise

  `attended.Day1[1] is 0`

Replace Day1 with Day2 and Day3 to get attendance on Days 2 and 3

Choices each day


Plenary session

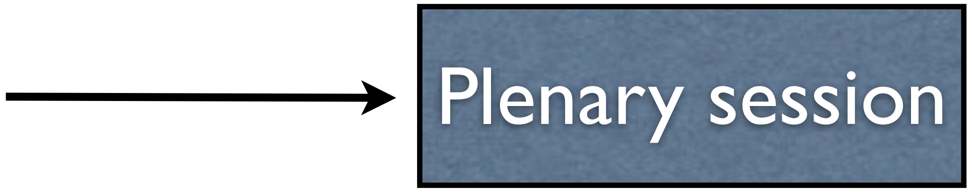
Issues panel

Forums

Workshop

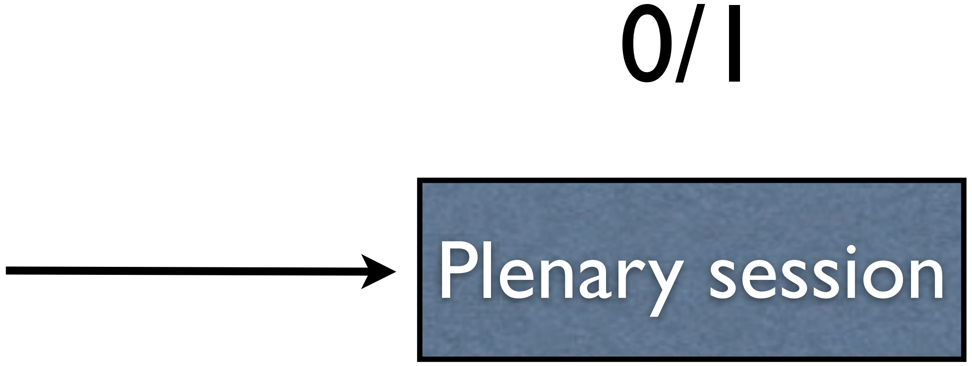
Research

Choices each day



Plenary session

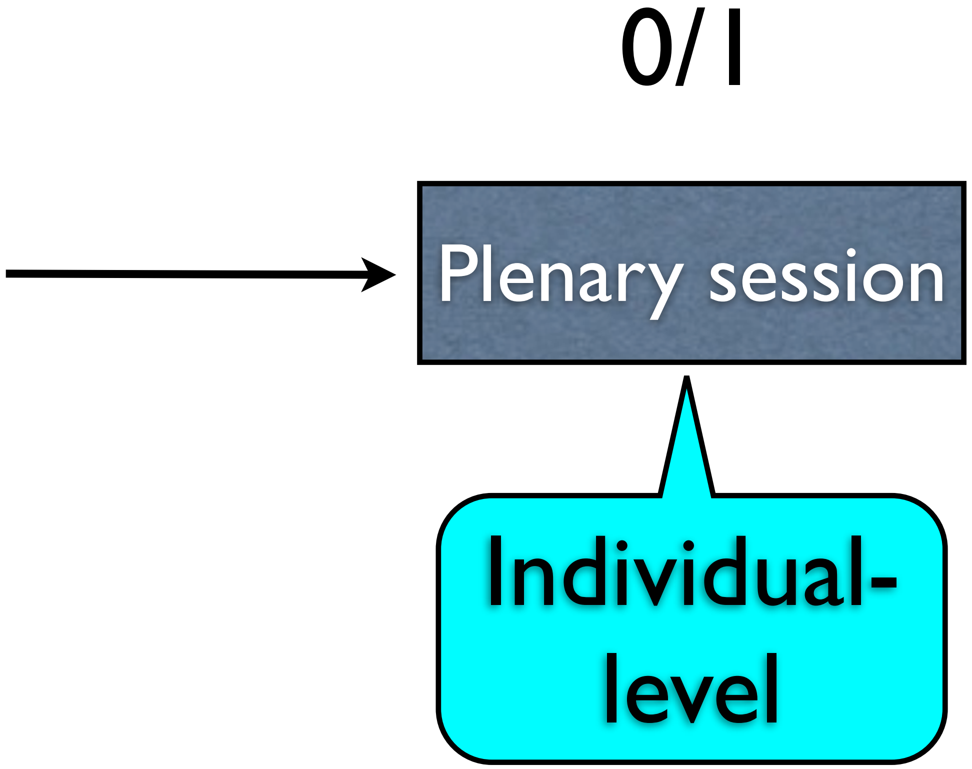
Choices each day



0/1

Plenary session

Choices each day

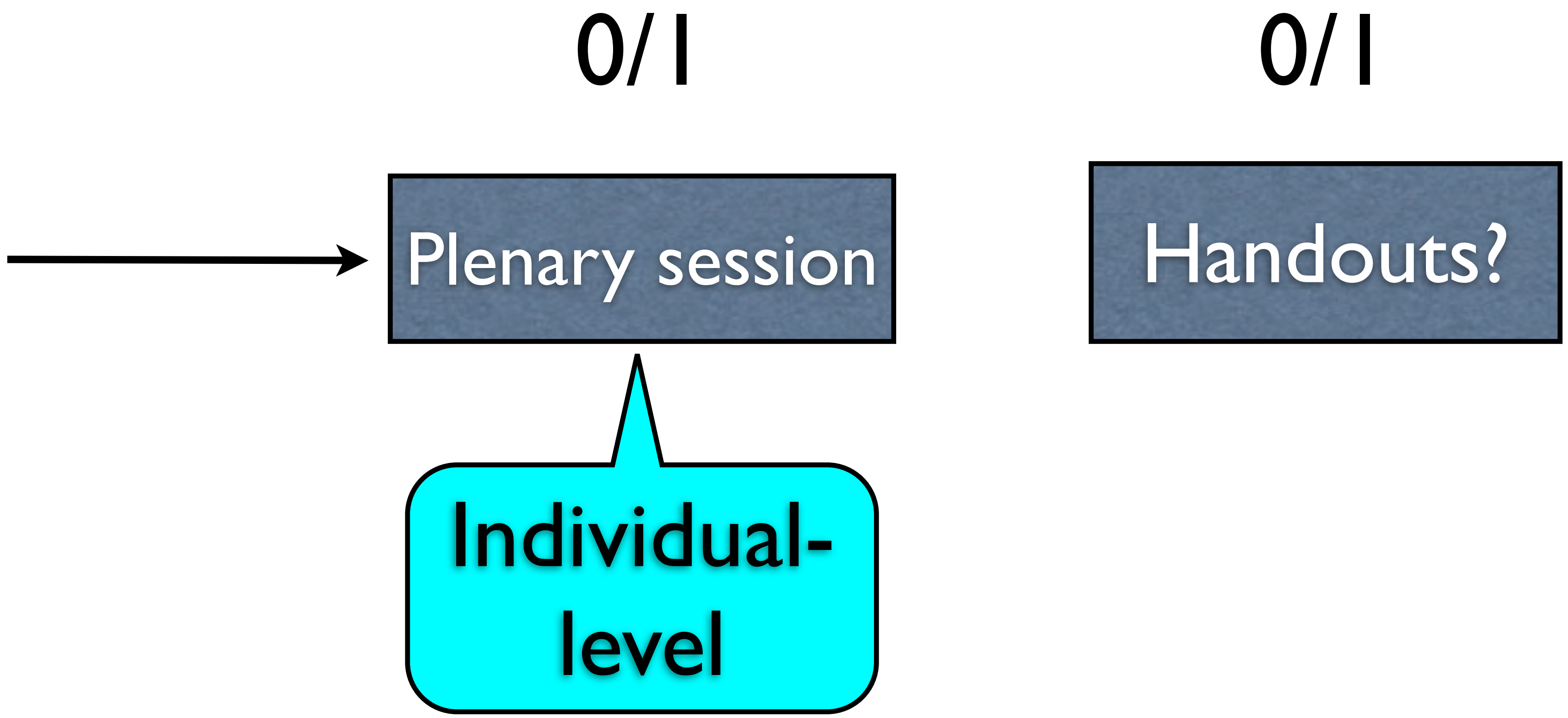


0/1

Plenary session

Individual-level

Choices each day



0/1

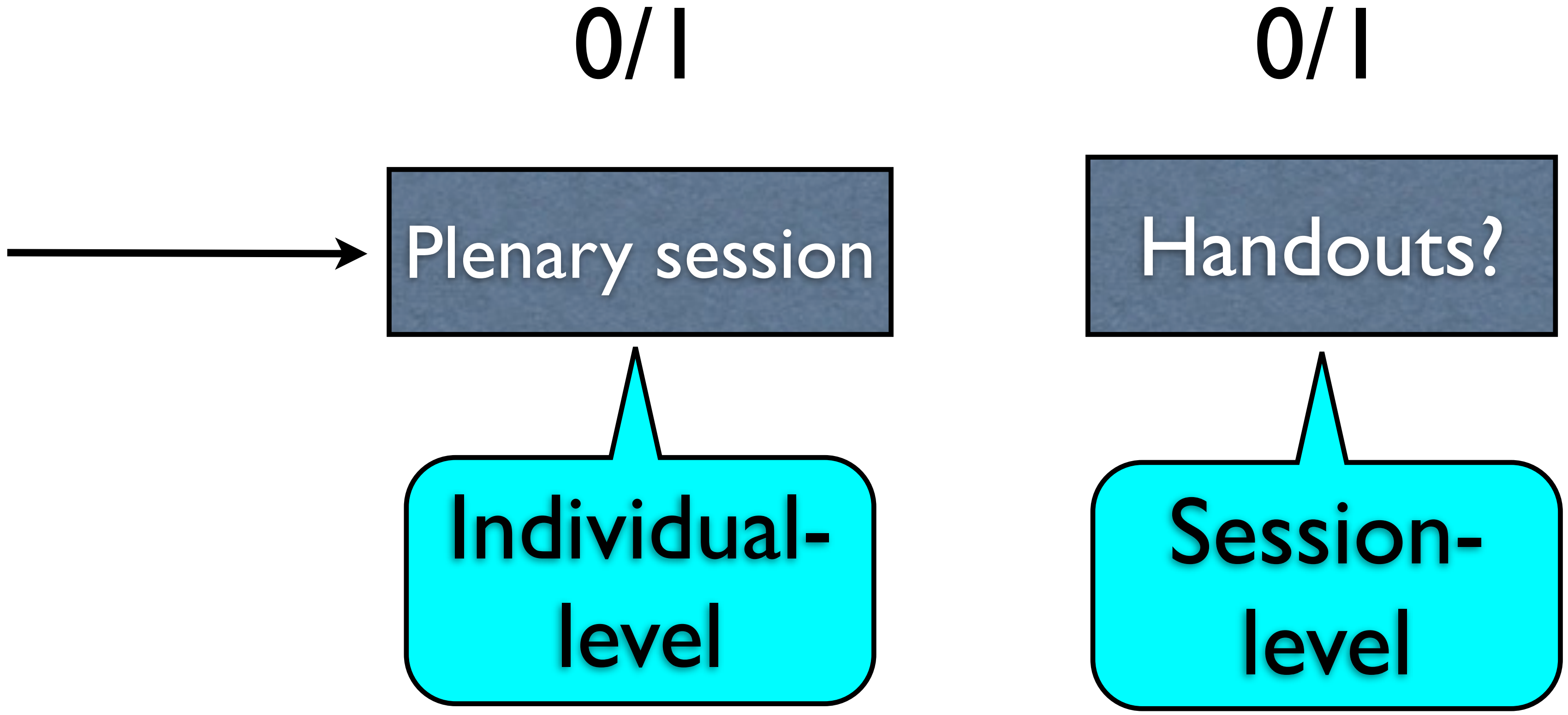
Plenary session

Individual-level

0/1

Handouts?

Choices each day



0/1

Plenary session

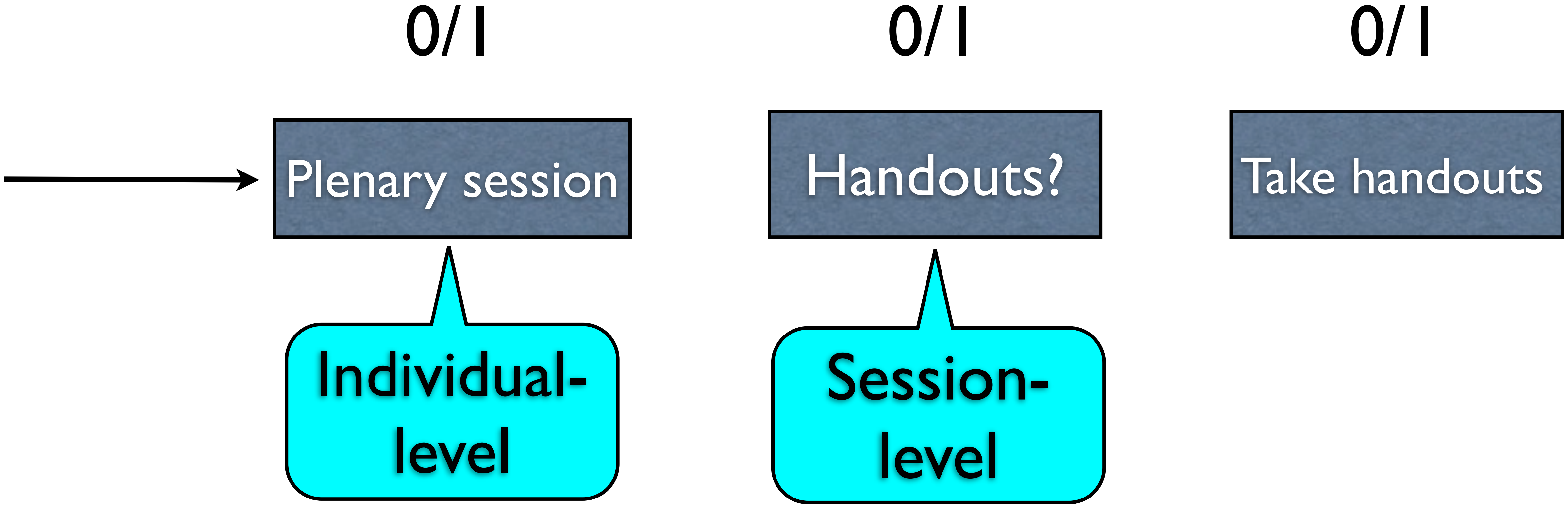
Individual-level

0/1

Handouts?

Session-level

Choices each day



0/1

Plenary session

Individual-level

0/1

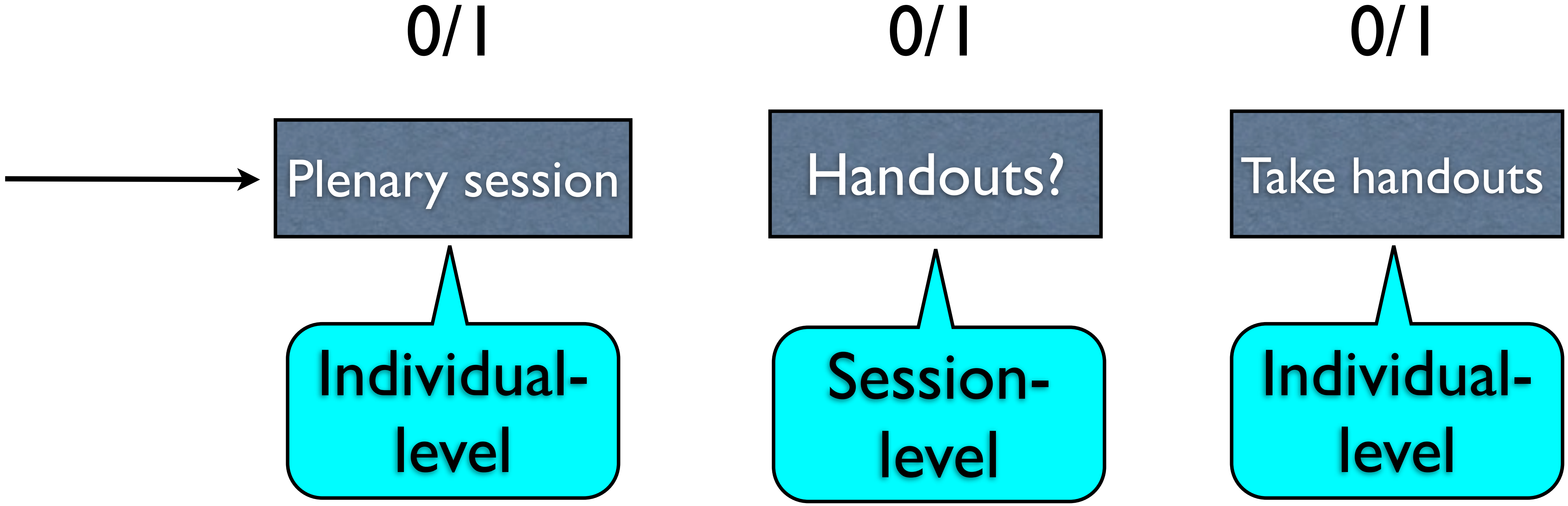
Handouts?

Session-level

0/1

Take handouts

Choices each day



0/1

Plenary session

Individual-level

0/1

Handouts?

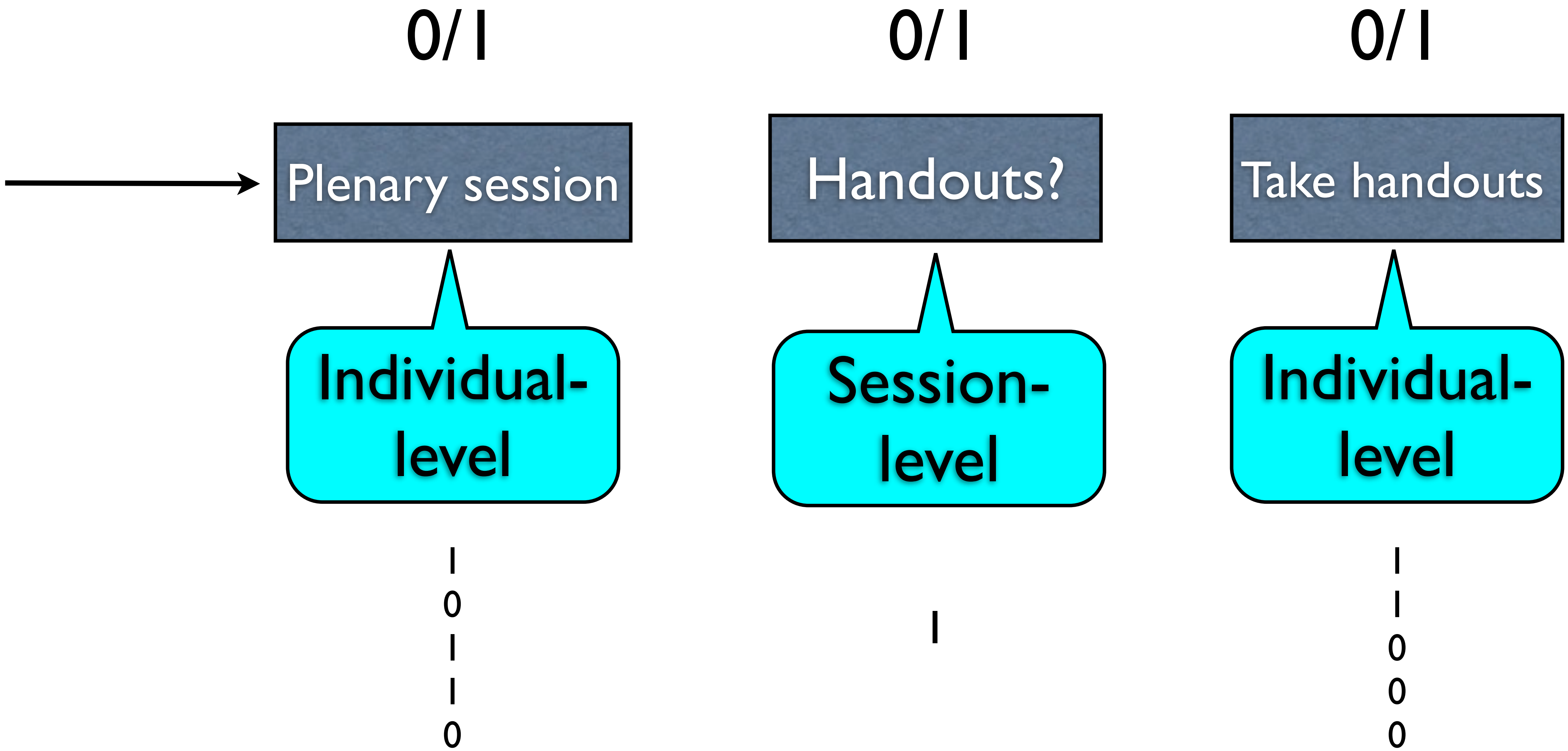
Session-level

0/1

Take handouts

Individual-level

Choices each day



0/1

Plenary session

Individual-level

1
0
1
1
0

0/1

Handouts?

Session-level

1

0/1

Take handouts

Individual-level

1
1
0
0
0

Choices

	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	Individual-level	Individual-level	Session-level	Individual-level
	1 1 1 0 1	1 0 1 1 0	1	1 1 0 0 0
Prob	0.8	0.4	0.95	0.5

Choices

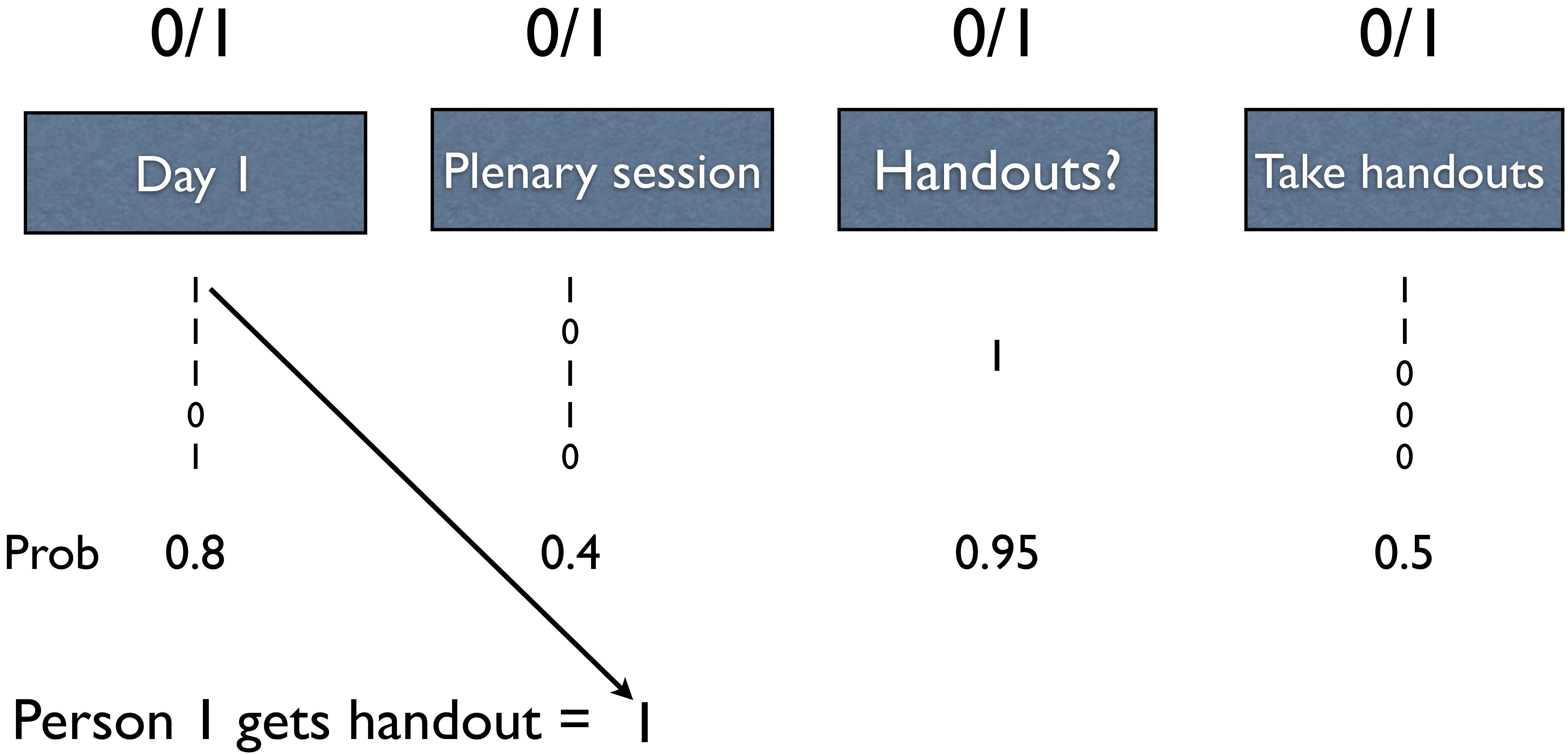
	Day 1	Plenary session	Handouts?	Take handouts
	0/1	0/1	0/1	0/1
	1 1 1 0 1	1 0 1 1 0	1	1 1 0 0 0
Prob	0.8	0.4	0.95	0.5

Choices

	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	1 1 1 0 1	1 0 1 1 0	1	1 1 0 0 0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout =

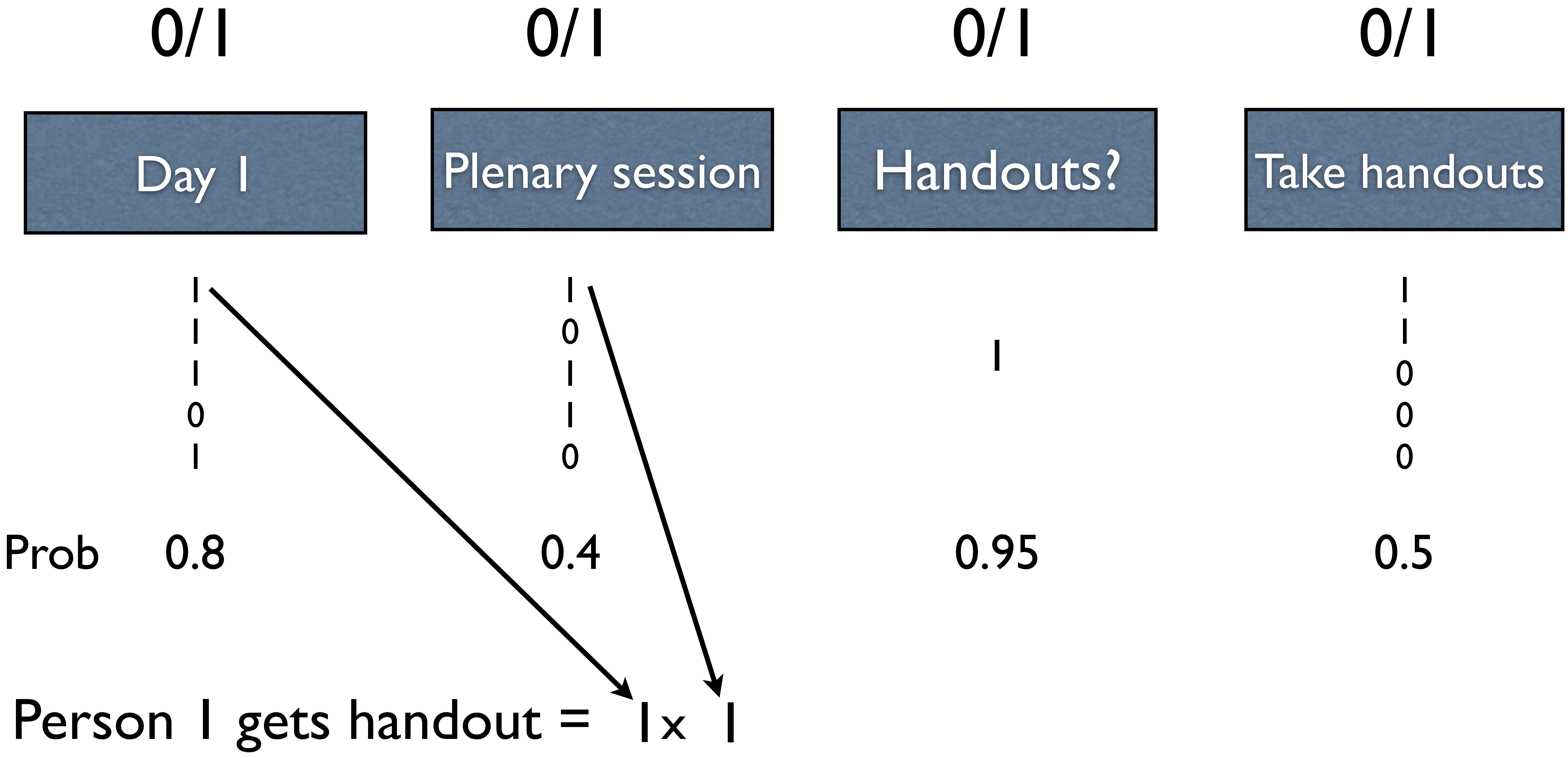
Choices



	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	1	1	1	1
	1	0		1
	1	1		0
	0	1		0
	1	0		0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout = 1

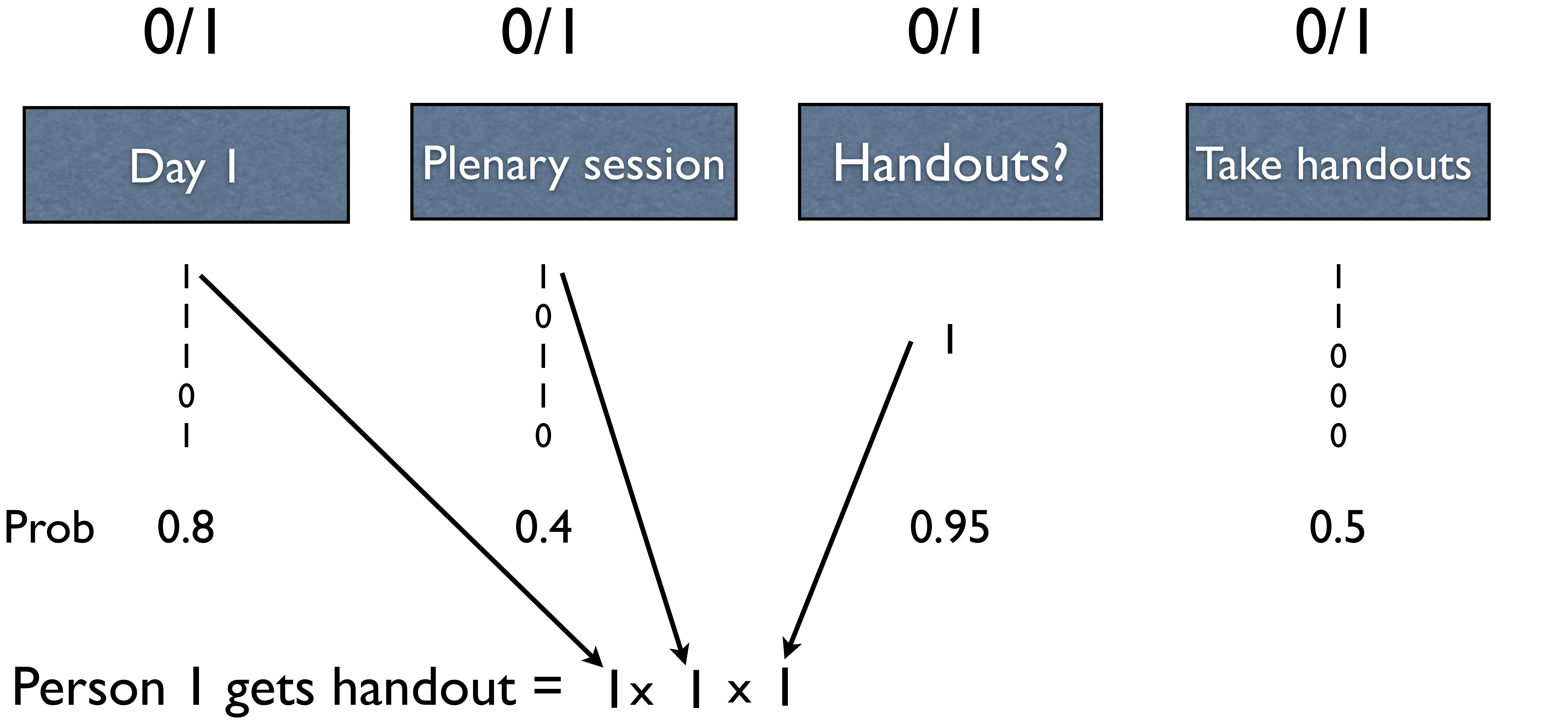
Choices



	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	1 1 1 0 1	1 0 1 1 0	1	1 1 0 0 0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout = 1x 1

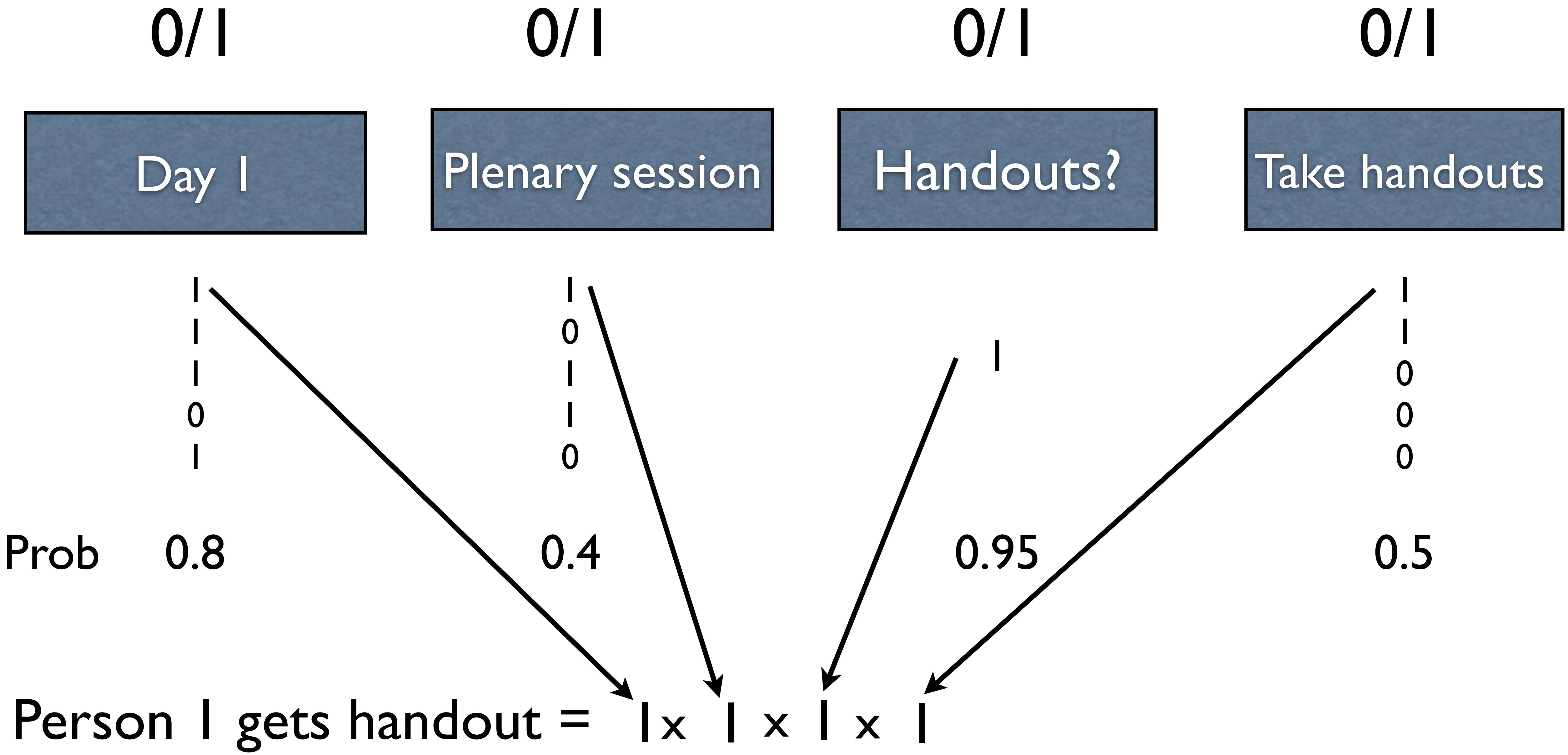
Choices



	Day 1	Plenary session	Handouts?	Take handouts
	0/1	0/1	0/1	0/1
	1	1	1	1
	1	0		1
	1	1		0
	0	1		0
	1	0		0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout = 1x 1 x 1

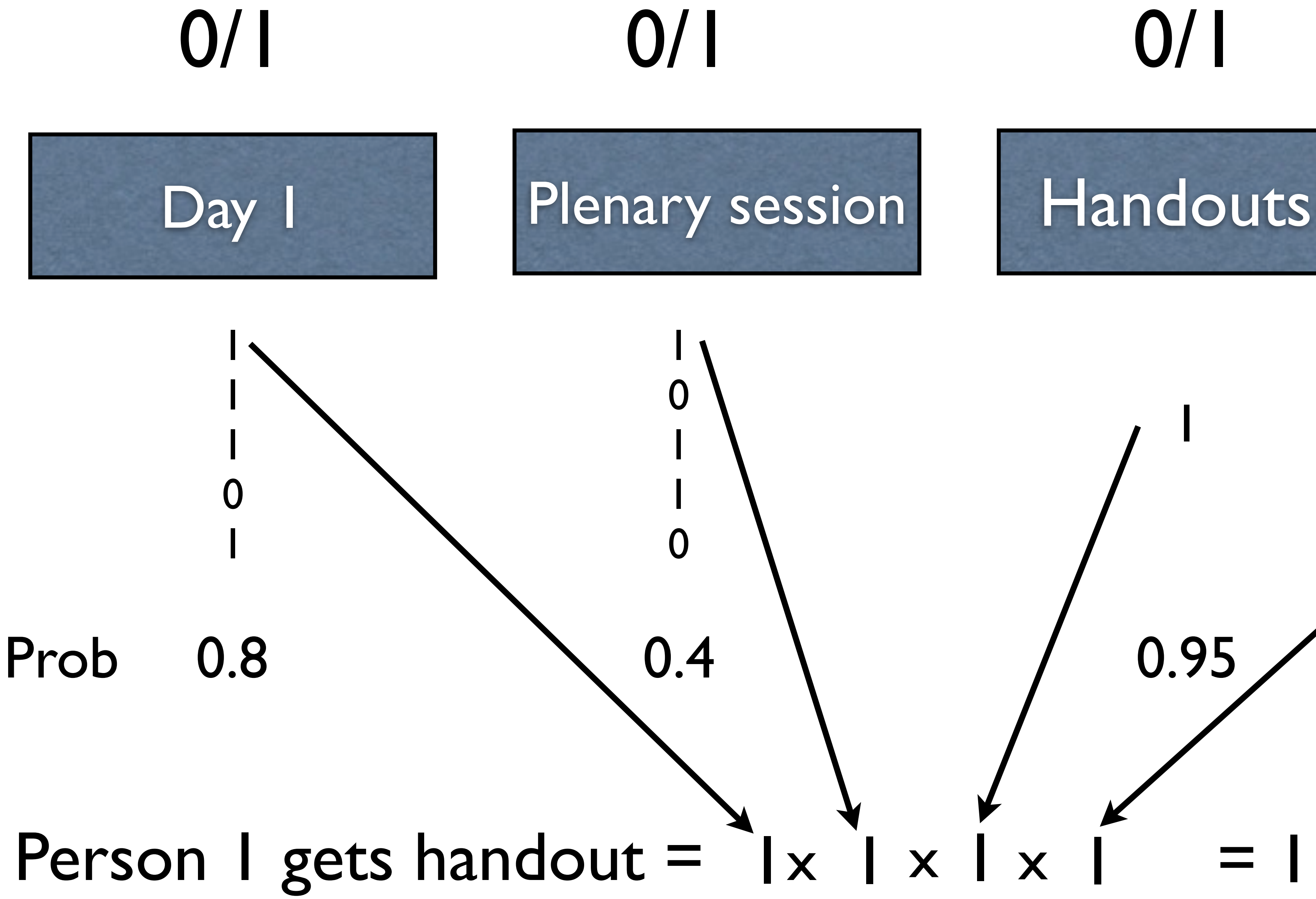
Choices



	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	1	1		1
	1	0	1	1
	1	1		0
	0	1		0
	1	0		0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout = 1x 1 x 1 x 1

Choices



0/1 0/1 0/1 0/1

Day 1

Plenary session

Handouts?

Take handouts

1
1
1
0
1

1
0
1
1
0

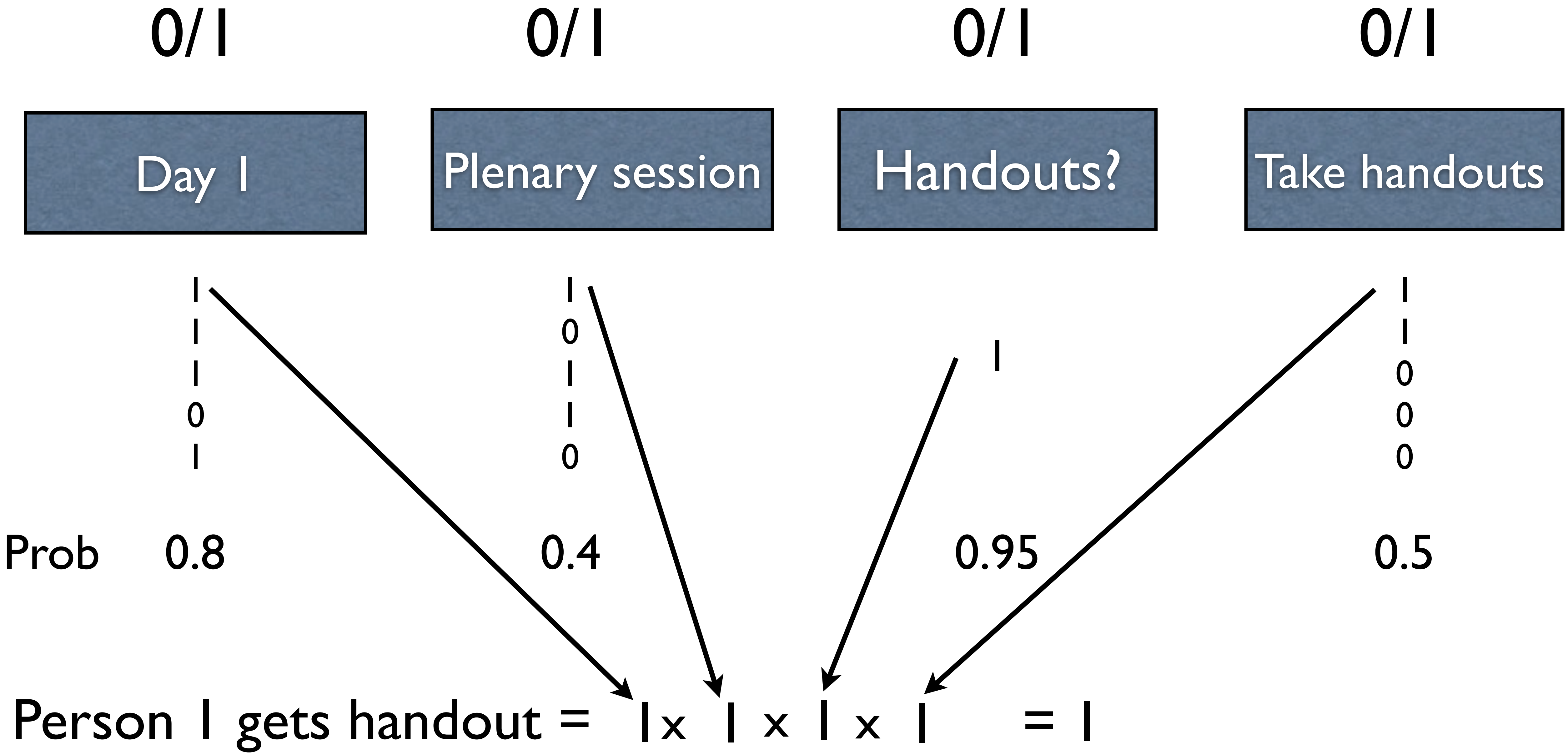
1

1
1
0
0
0

Prob 0.8 0.4 0.95 0.5

Person 1 gets handout = 1 x 1 x 1 x 1 = 1

Choices

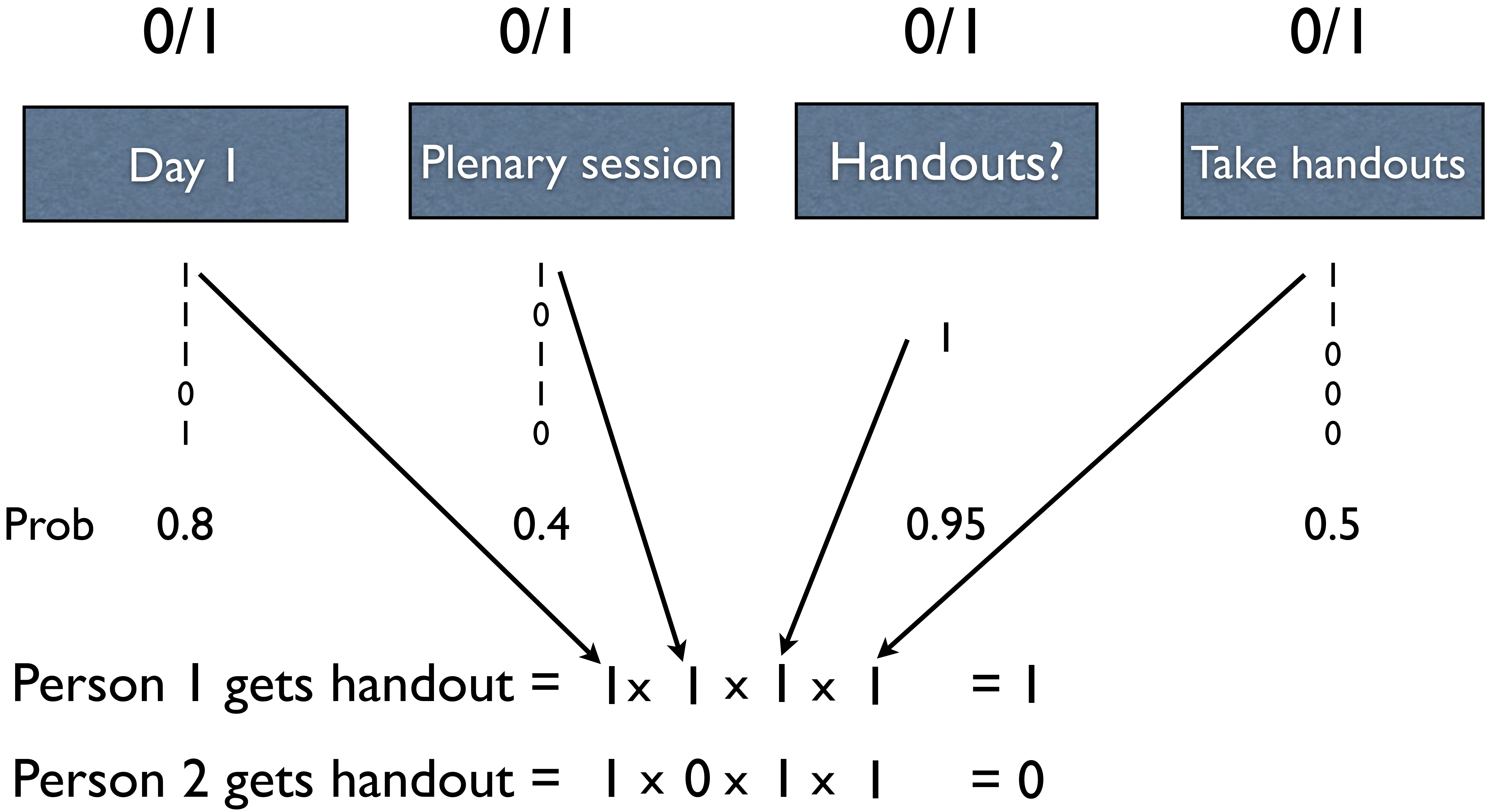


	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	1 1 1 0 1	1 0 1 1 0	1	1 1 0 0 0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout = 1x 1 x 1 x 1 = 1

Person 2 gets handout =

Choices

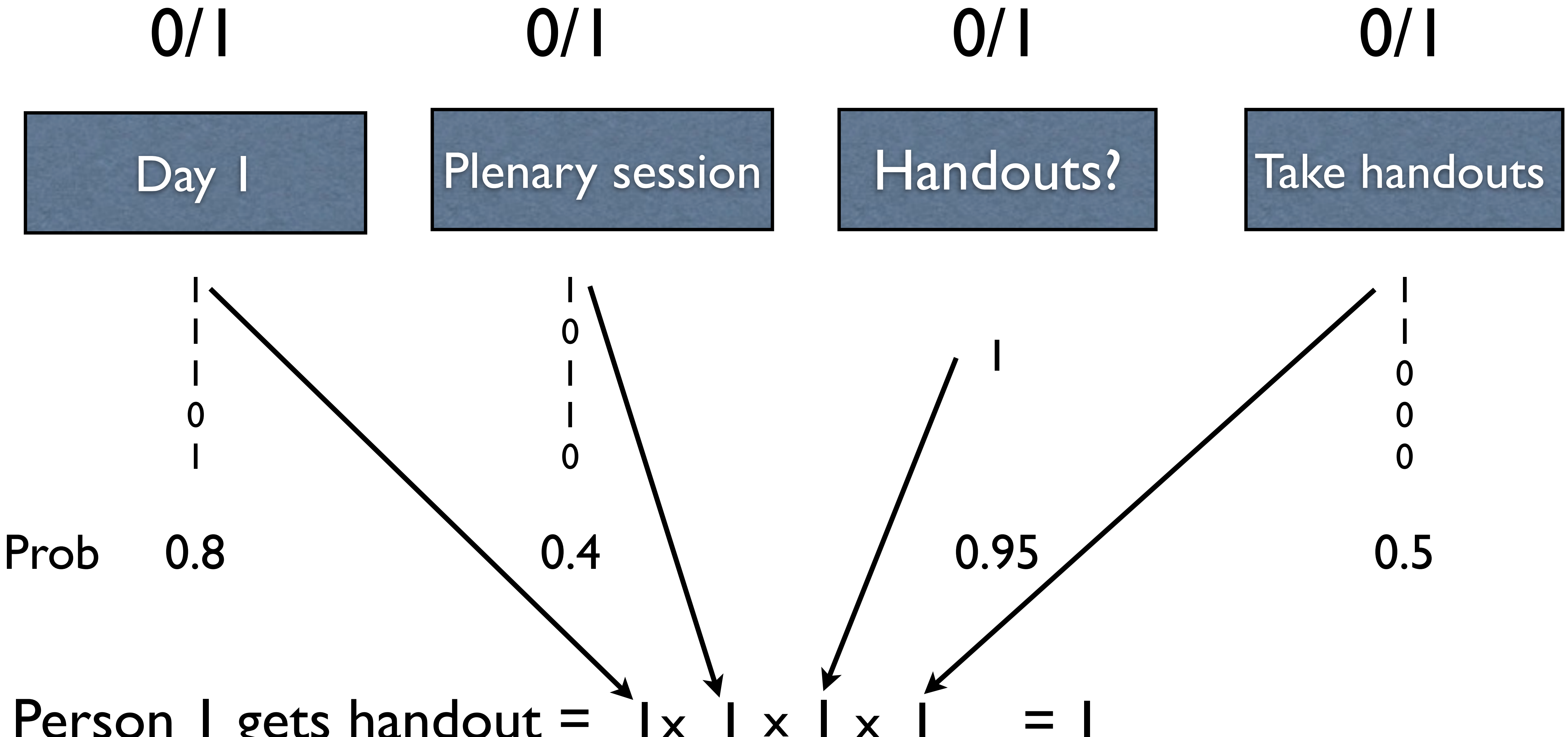


	0/1	0/1	0/1	0/1
	Day 1	Plenary session	Handouts?	Take handouts
	1	1		1
	1	0	1	1
	1	1		0
	0	1		0
	1	0		0
Prob	0.8	0.4	0.95	0.5

Person 1 gets handout = 1 x 1 x 1 x 1 = 1

Person 2 gets handout = 1 x 0 x 1 x 1 = 0

Choices



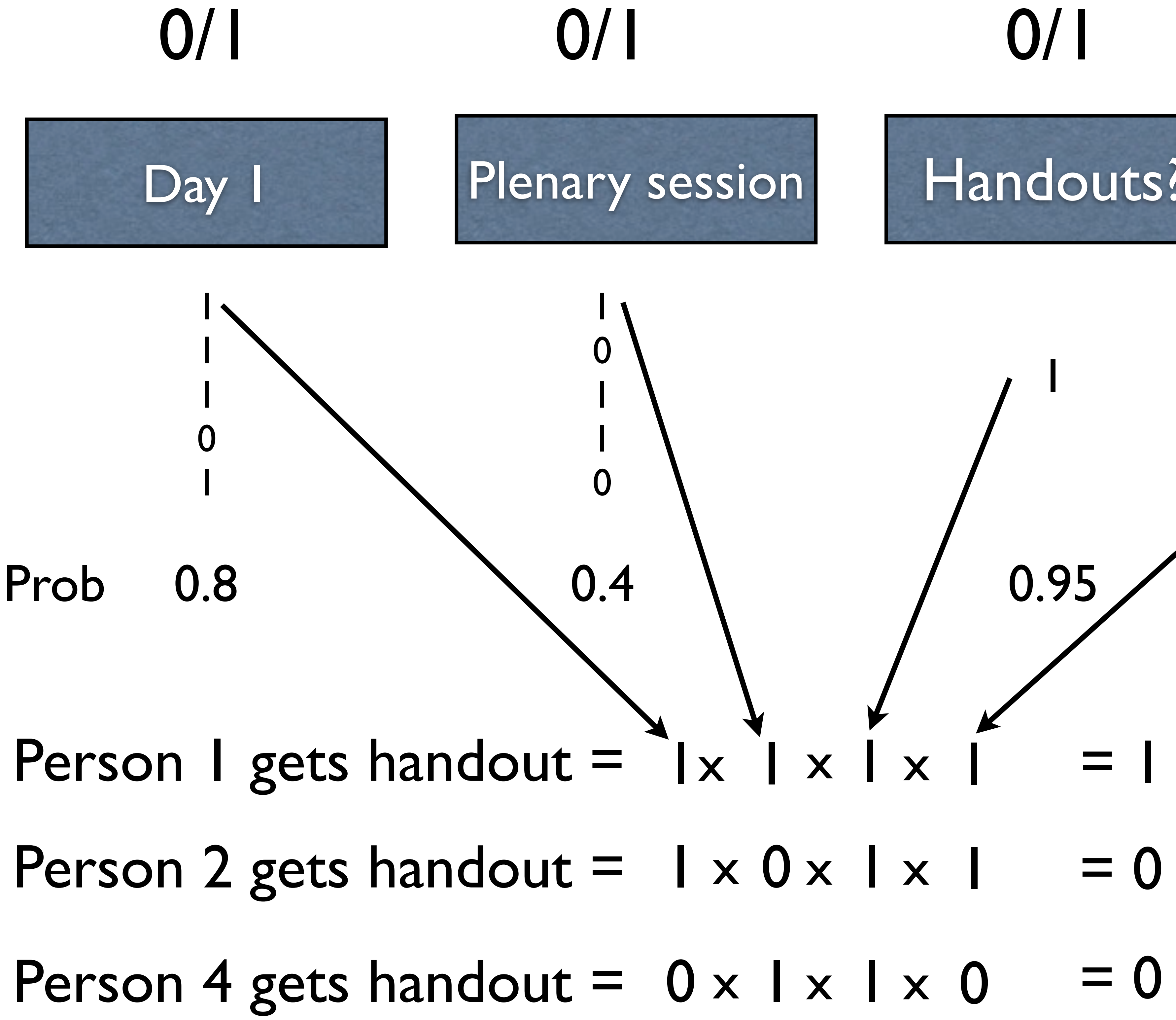
0/1	0/1	0/1	0/1
Day 1	Plenary session	Handouts?	Take handouts
1	1	1	1
1	0		1
1	1		0
0	1		0
1	0		0
Prob 0.8	0.4	0.95	0.5

Person 1 gets handout = 1x 1 x 1 x 1 = 1

Person 2 gets handout = 1 x 0 x 1 x 1 = 0

Person 4 gets handout = 0 x 1 x 1 x 0 = 0

Choices



0/1	0/1	0/1	0/1
Day 1	Plenary session	Handouts?	Take handouts
1 1 1 0 1	1 0 1 1 0	1	1 1 0 0 0
Prob 0.8	0.4	0.95	0.5

Person 1 gets handout = 1x 1 x 1 x 1 = 1

Person 2 gets handout = 1 x 0 x 1 x 1 = 0

Person 4 gets handout = 0 x 1 x 1 x 0 = 0

All conditions need to be met!!

Getting handout at Day 1 Plenary session

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

```
attended.Plenary.Session = sample.int(2, size=2500,
                    replace=T,
                    prob=c(1-p.Plenary,p.Plenary))-1
```

```
handout.available.Plenary = sample.int(2, size=1,
    prob=c(1-p.Plenary.handout,p.Plenary.handout))-1
```

```
take.handout = sample.int(2, size=2500,
          replace=T,
          prob=c(1-p.take.handout,p.take.handout))-1
```

Getting handout at Day 1 Plenary session

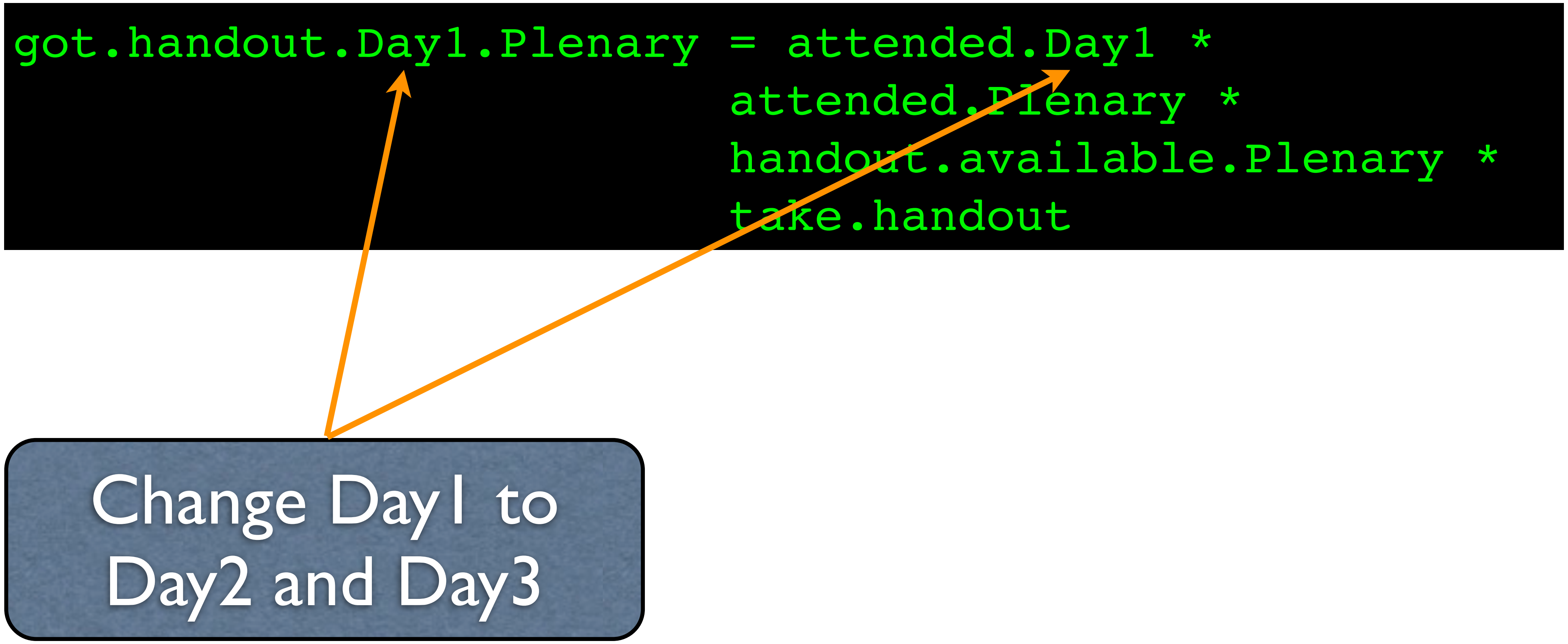
```
got.handout.Day1.Plenary = attended.Day1 *
                         attended.Plenary *
                         handout.available.Plenary *
                         take.handout
```

Getting handout at Day 1 Plenary session

```
got.handout.Day1.Plenary = attended.Day1 *
                           attended.Plenary *
                           handout.available.Plenary *
                           take.handout
```

1
0
1
0
0
1
1

Getting handout at Day 1 Plenary session

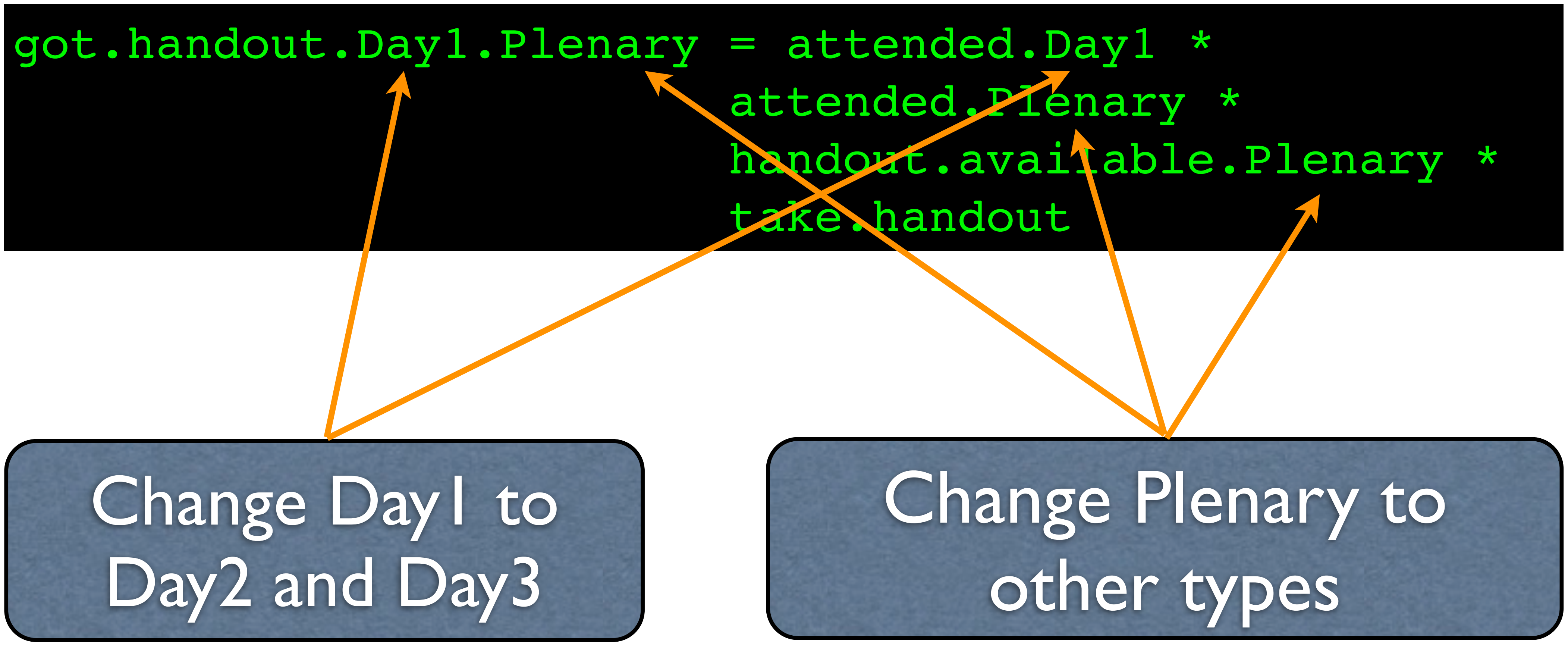


```
got.handout.Day1.Plenary = attended.Day1 *
                           attended.Plenary *
                           handout.available.Plenary *
                           take.handout
```

Change Day1 to Day2 and Day3

1
0
1
0
0
1
1

Getting handout at Day 1 Plenary session



```
got.handout.Day1.Plenary = attended.Day1 *
                         attended.Plenary *
                         handout.available.Plenary *
                         take.handout
```

1
0
1
0
0
1
1

Create functions for repetitive tasks

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

Make this generic

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

Make this generic

```
attended.Day = function(p.Day, size=2500){
    x = sample.int(2, size=size, replace=T,
                   prob=c(1-p.Day,p.Day))
    return(x)
}
```

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

Make this generic

```
attended.Day = function(p.Day, size=2500){
    x = sample.int(2, size=size, replace=T,
                   prob=c(1-p.Day,p.Day))
    return(x)
}
```

Replace with the probabilities for each day

```
attended.Day1 = sample.int(2, size=2500,
                           replace=T,
                           prob=c(1-p.Day1,p.Day1))-1
```

Make this generic

```
attended.Day = function(p.Day, size=2500){
    x = sample.int(2, size=size, replace=T,
                   prob=c(1-p.Day,p.Day))
    return(x)
}
```

Replace with the probabilities for each day

So

```
attended.Day1 = attended.Day(p.Day1)
attended.Day2 = attended.Day(p.Day2)
attended.Day3 = attended.Day(p.Day3)
attended.Plenary.Session = sample.int(2, size=2500,
                    replace=T,
                    prob=c(1-p.Plenary,p.Plenary))-1
```

```
attended.session = function(p.session, size=2500){
    x = sample.int(2, size=size,
                    replace=T,
                    prob=c(1-p.session,p.session))-1
    return(x)
}
```

```
attended.Plenary.Session = sample.int(2, size=2500,
                    replace=T,
                    prob=c(1-p.Plenary,p.Plenary))-1
```

```
attended.session = function(p.session, size=2500){
    x = sample.int(2, size=size,
                    replace=T,
                    prob=c(1-p.session,p.session))-1
    return(x)
}
```

Replace with session-specific probability

```
attended.Plenary.Session = sample.int(2, size=2500,
                    replace=T,
                    prob=c(1-p.Plenary,p.Plenary))-1
```

```
attended.session = function(p.session, size=2500){
    x = sample.int(2, size=size,
                    replace=T,
                    prob=c(1-p.session,p.session))-1
    return(x)
}
```

Replace with session-specific probability

```
attended.Plenary.Session = attended.session(p.Plenary)
attended.Forum           = attended.session(p.Forum)
attended.Research.Session = attended.session(p.Research)
attended.Workshop        = attended.session(p.Workshop)
attended.Issues.Panel    = attended.session(p.Issues)
```

```
is.handout.available = function(p.handout.session, size=1){
    x = sample.int(2, size=size,
              replace=T,
              prob=c(1-p.handout.session,p.handout.session))-1
    return(x)
}
```

```
is.handout.available = function(p.handout.session, size=1){
    x = sample.int(2, size=size,
              replace=T,
              prob=c(1-p.handout.session,p.handout.session))-1
    return(x)
}
```

```
is.handout.available = function(p.handout.session, size=1){
    x = sample.int(2, size=size,
              replace=T,
              prob=c(1-p.handout.session,p.handout.session))-1
    return(x)
}
```

```
taking.handout = function(p = p.take.handout, size=2500){
    x = sample.int(2, size=size, replace=T,
                   prob = c(1-p, p)) -1
    return(x)
}
```

```
is.handout.available = function(p.handout.session, size=1){
    x = sample.int(2, size=size,
              replace=T,
              prob=c(1-p.handout.session,p.handout.session))-1
    return(x)
}
```

```
taking.handout = function(p = p.take.handout, size=2500){
    x = sample.int(2, size=size, replace=T,
                   prob = c(1-p, p)) -1
    return(x)
}
```

Note both have default values

```
is.handout.available = function(p.handout.session, size=1){
    x = sample.int(2, size=size,
                  replace=T,
                  prob=c(1-p.handout.session,p.handout.session))-1
    return(x)
}
```

```
taking.handout = function(p = p.take.handout, size=2500){
    x = sample.int(2, size=size, replace=T,
                       prob = c(1-p, p)) -1
    return(x)
}
```

Note both have default values

Why do this?

We have to re-generate whether someone took a handout for each session of each day

Getting handout at Day 1 Plenary session

```
got.handout.Day1.Plenary = attended.Day1 *
                           attended.Plenary *
                           handout.available.Plenary *
                           take.handout
```

↕

```
got.handout.Day1.Plenary =
     attended.Day(p.Day1) *
     attended.session(p.Plenary) *
     is.handout.available(p.handout.Plenary) *
     taking.handout()
```

Getting handout at Day 1 Plenary session

```
got.handout.Day1.Plenary = attended.Day1 *
                           attended.Plenary *
                           handout.available.Plenary *
                           take.handout
```

```
got.handout.Day1.Plenary =
     attended.Day(p.Day1) *
     attended.session(p.Plenary) *
     is.handout.available(p.handout.Plenary) *
     taking.handout()
```

Can change to other session types

Getting handout on Day 1

```
got.handout.Day1 = cbind(got.handout.Day1.Plenary,
    got.handout.Day1.Issue,
    got.handout.Day1.Research1,
    got.handout.Day1.Research2,
    got.handout.Day1.Workshop,
    got.handout.Day1.Forum)
```

Getting handout on Day 1

```
got.handout.Day1 = cbind(got.handout.Day1.Plenary,
    got.handout.Day1.Issue,
    got.handout.Day1.Research1,
    got.handout.Day1.Research2,
    got.handout.Day1.Workshop,
    got.handout.Day1.Forum)
```

These are generated using the same code, but are different in value due to the random number generation

Getting handout on Day 1

```
got.handout.Day1 = cbind(got.handout.Day1.Plenary,
    got.handout.Day1.Issue,
    got.handout.Day1.Research1,
    got.handout.Day1.Research2,
    got.handout.Day1.Workshop,
    got.handout.Day1.Forum)
```

These are generated using the same code, but are different in value due to the random number generation

```
got.handout.Day1.Research1 =
        attended.Day(p.Day1) * attended.session(p.Research) *
        is.handout.available(p.handout.Research) *
        taking.handout()
```

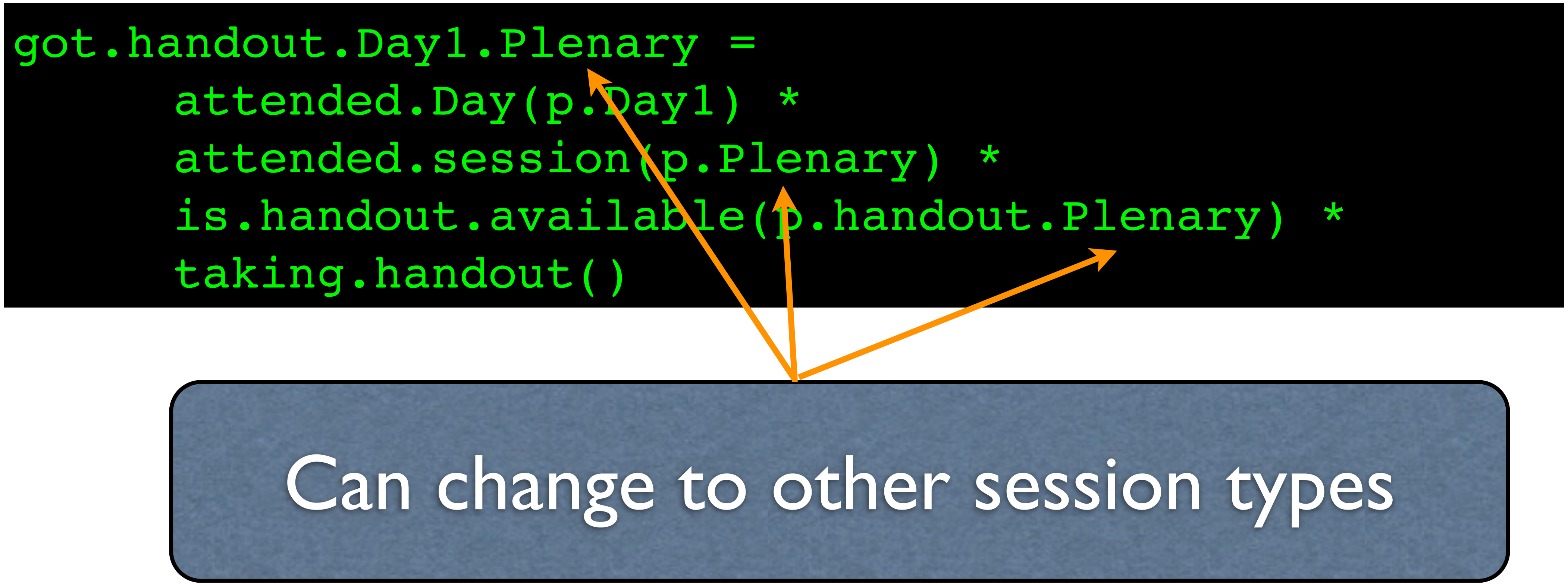
```
got.handout.Day1.Research2 =
        attended.Day(p.Day1) * attended.session(p.Research) *
        is.handout.available(p.handout.Research) *
        taking.handout()
```

Handouts on Day 1

`got.handout.Day1`

Plenary session	Issue Panel	Research	Research	Workshop	Forum
1	0	1	1	0	0
0	0	1	1	0	0
0	1	0	1	1	0
1	0	0	0	0	1
1	1	0	1	1	0
0	1	1	1	0	1
1	1	0	0	0	0
0	0	1	0	1	0
1	0	0	1	0	0
1	1	0	1	0	1

Programmatic generation of got.handout.Day1



```
got.handout.Day1.Plenary =
      attended.Day(p.Day1) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

Can change to other session types

```
got.handout.Day1.Plenary =
      attended.Day(p.Day1) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

```
got.handout.Day1.Plenary =
      attended.Day(p.Day1) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

```
probs.for.Day1.sessions = c(p.Plenary, p.Issue,
                            p.Research, p.Research,
                            p.Workshop, p.Forum)

probs.for.getting.handouts.at.Day1.sessions =
    c(p.handout.Plenary, p.handout.Issue,
      p.handout.Research, p.handout.Research,
      p.handout.Workshop, p.handout.Forum)
```

```
got.handout.Day1.Plenary =
      attended.Day(p.Day1) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

```
probs.for.Day1.sessions = c(p.Plenary, p.Issue,
                            p.Research, p.Research,
                            p.Workshop, p.Forum)

probs.for.getting.handouts.at.Day1.sessions =
    c(p.handout.Plenary, p.handout.Issue,
      p.handout.Research, p.handout.Research,
      p.handout.Workshop, p.handout.Forum)
```

```
got.handout.Day1 = matrix(0, nrow=2500, ncol=6)
for(i in 1:6){
  got.handout.Day1[,i] =
    attended.Day(p.Day1)*
    attended.session(probs.for.Day1.session[i])*
    is.handout.available(
    probs.for.getting.handouts.at.Day1.sessions[i]) *
    taking.handout()
}
```

```
got.handout.Day2.Plenary =
      attended.Day(p.Day2) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

```
got.handout.Day2.Plenary =
      attended.Day(p.Day2) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

```
probs.for.Day2.sessions = c(p.Research, p.Plenary,
                            p.Issue, p.Workshop,
                            p.Workshop, p.Forum)

probs.for.getting.handouts.at.Day1.sessions =
    c(p.handout.Research, p.handout.Plenary,
      p.handout.Issue, p.handout.Workshop,
      p.handout.Workshop, p.handout.Forum)
```

```
got.handout.Day2.Plenary =
      attended.Day(p.Day2) *
      attended.session(p.Plenary) *
      is.handout.available(p.handout.Plenary) *
      taking.handout()
```

```
probs.for.Day2.sessions = c(p.Research, p.Plenary,
                            p.Issue, p.Workshop,
                            p.Workshop, p.Forum)

probs.for.getting.handouts.at.Day1.sessions =
    c(p.handout.Research, p.handout.Plenary,
      p.handout.Issue, p.handout.Workshop,
      p.handout.Workshop, p.handout.Forum)
```

```
got.handout.Day2 = matrix(0, nrow=2500, ncol=6)
for(i in 1:6){
  got.handout.Day2[,i] =
    attended.Day(p.Day2)*
    attended.session(probs.for.Day2.session[i])*
    is.handout.available(
    probs.for.getting.handouts.at.Day2.sessions[i]) *
    taking.handout()
}
```

Leveraging programming again

```
probs.Day = c(p.Day1,p.Day2,p.Day3)
probs.for.sessions =
        list(probs.for.Day1.sessions,
             probs.for.Day2.sessions,
             probs.for.Day3.sessions)

probs.for.getting.handouts.at.sessions =
     list(probs.for.getting.handouts.at.Day1.sessions,
          probs.for.getting.handouts.at.Day1.sessions,
          probs.for.getting.handouts.at.Day1.sessions)

got.handouts = vector(‘list’,3)
for(day in 1:3){
  got.handouts[[day]] = matrix(0, nrow=2500,
    ncol=length(probs.for.sessions[[day]]))
    for(session in 1:length(probs.for.sessions[[day]]){
      got.handouts[[day]][,session] =
        attended.Day(probs.Day[i])*
      attended.session(probs.for.sessions[[day]][session])*
      is.handout.available(
       probs.for.getting.handouts.at.sessions[[day]][session]) *
      taking.handout()
    }
}
```

Final outcome is total number of handouts per person

We already have whether each person got a handout at each session of each day in `got.handouts`

```
total.handouts = rep(0,2500)
for(day in 1:3){
  total.handouts = total.handouts + rowSums(got.handouts[[day]])
}
```

Alternatively

```
x = sapply(got.handouts, rowSums) # gives 2500 x 3 matrix
total.handouts = rowSums(x)

# total.handouts = rowSums(sapply(got.handouts, rowSums))
```

Adding stochastic simulation

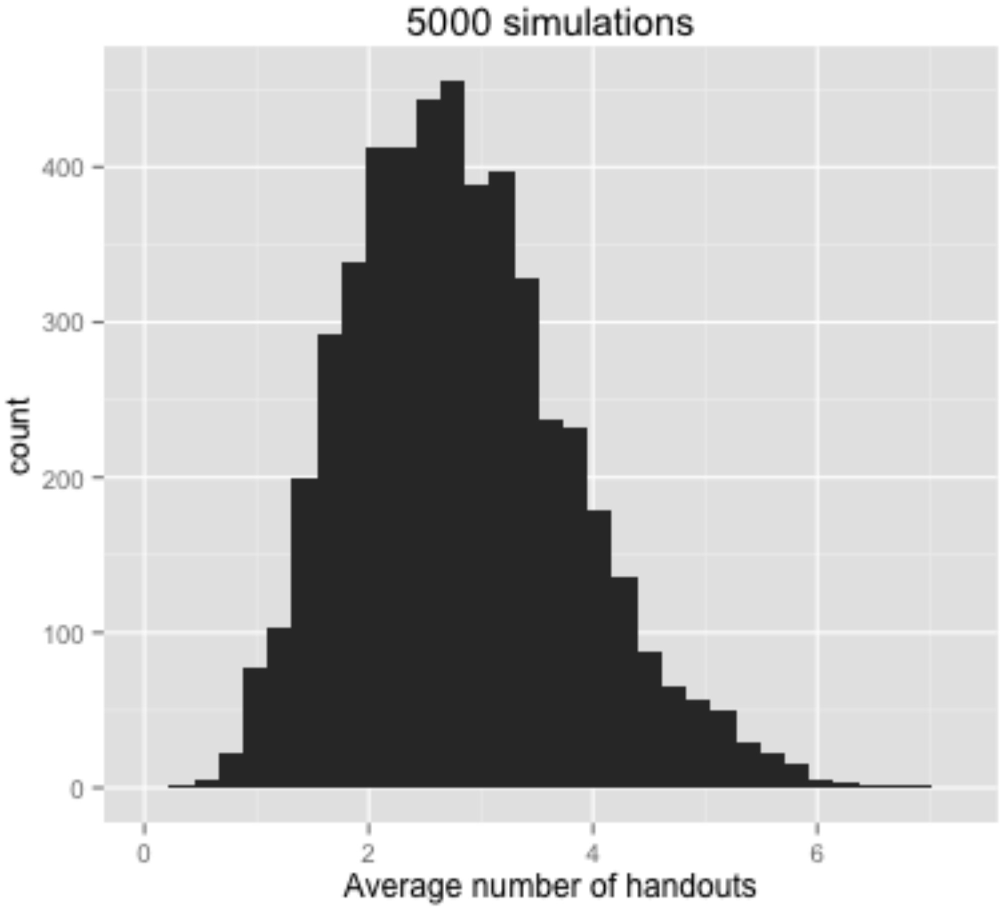
- Generate Beta-distributed numbers using `rbeta(N, a, b)`, where `a` and `b` are specified parameters (a/(a+b) = mean of distribution)

Adding stochastic simulation

```
p.Day1 = rbeta(1,8,2)
p.Day2 = rbeta(1,8,2)
p.Day3 = rbeta(1,3,2)
p.Plenary = rbeta(1,4,6)
...
```

Adding stochastic simulation

Now loop through `Nsim` times to get simulated average distribution



Approx 25s on single thread Mac, 2.4 GHz Core2 Duo

Parallelize computation

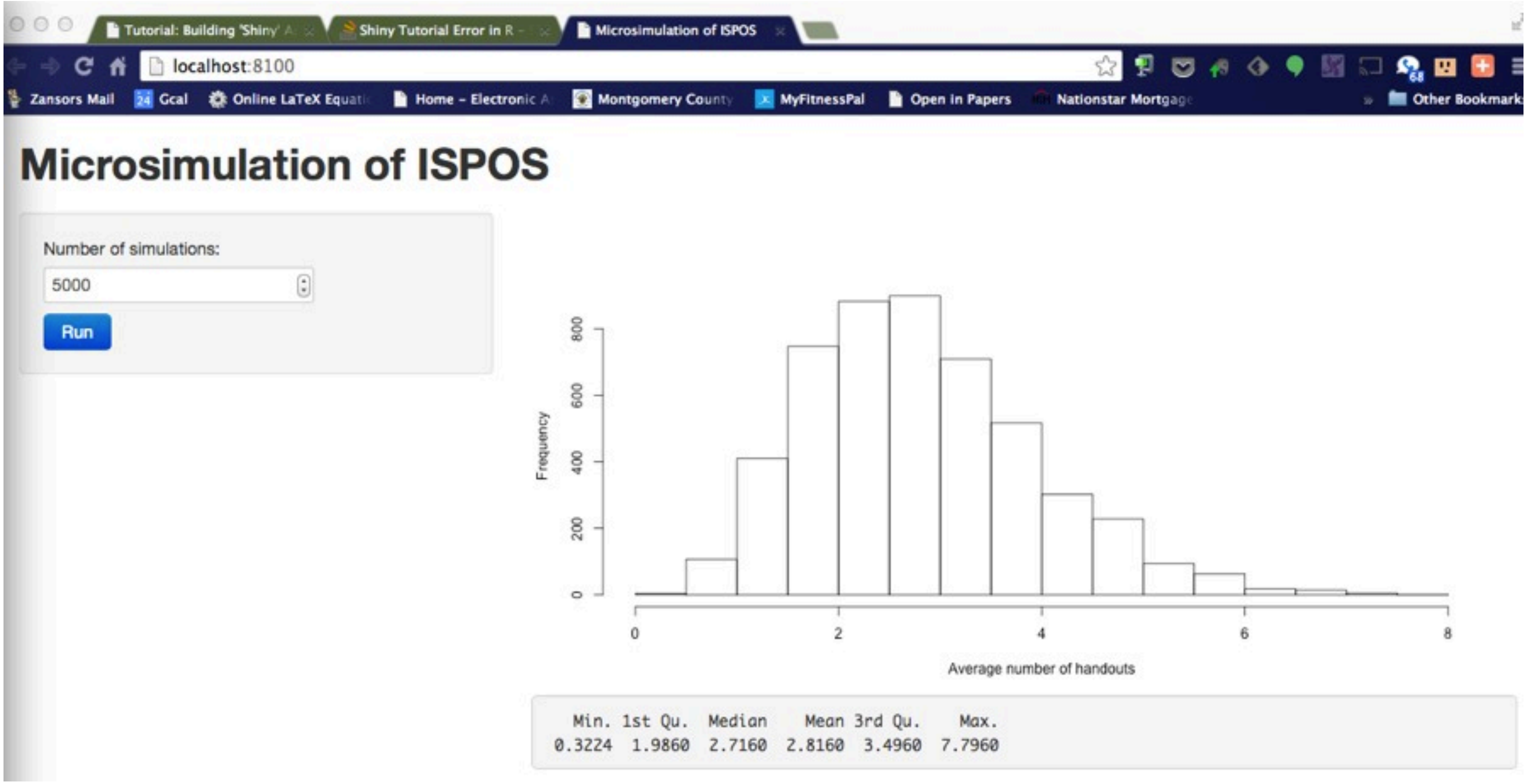
Possible in this problem

The packages `foreach` and `doParallel` allow this

```
sims.parallel <- function(Nsim,N){
  require(foreach)
  require(doParallel) # for Windows/Mac/Linux
  cl <- makeCluster(2)
  registerDoParallel(cl)
  # On windows replace previous 2 lines with
  # registerDoParallel(cores=2)
  output <- foreach(icount(Nsim),
                    .export=c('getbin','get.Handouts','probs','get.TotHandouts'),
                    .combine=cbind) %dopar% {
    get.TotHandouts(N)
  }
  stopCluster(cl)
  return(output)
}
```

Deployment

- The package Shiny allows deploying this using a web browser
  - Locally, or
  - From your own web server
  - See ui.R and server.R. Deploy locally using `runApps(‘.’)`
- `http://rstudio.github.io/shiny/tutorial/`



Microsimulation of ISPOS

Number of simulations: 5000

```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.3224  1.9860  2.7160  2.8160  3.4960  7.7960
```