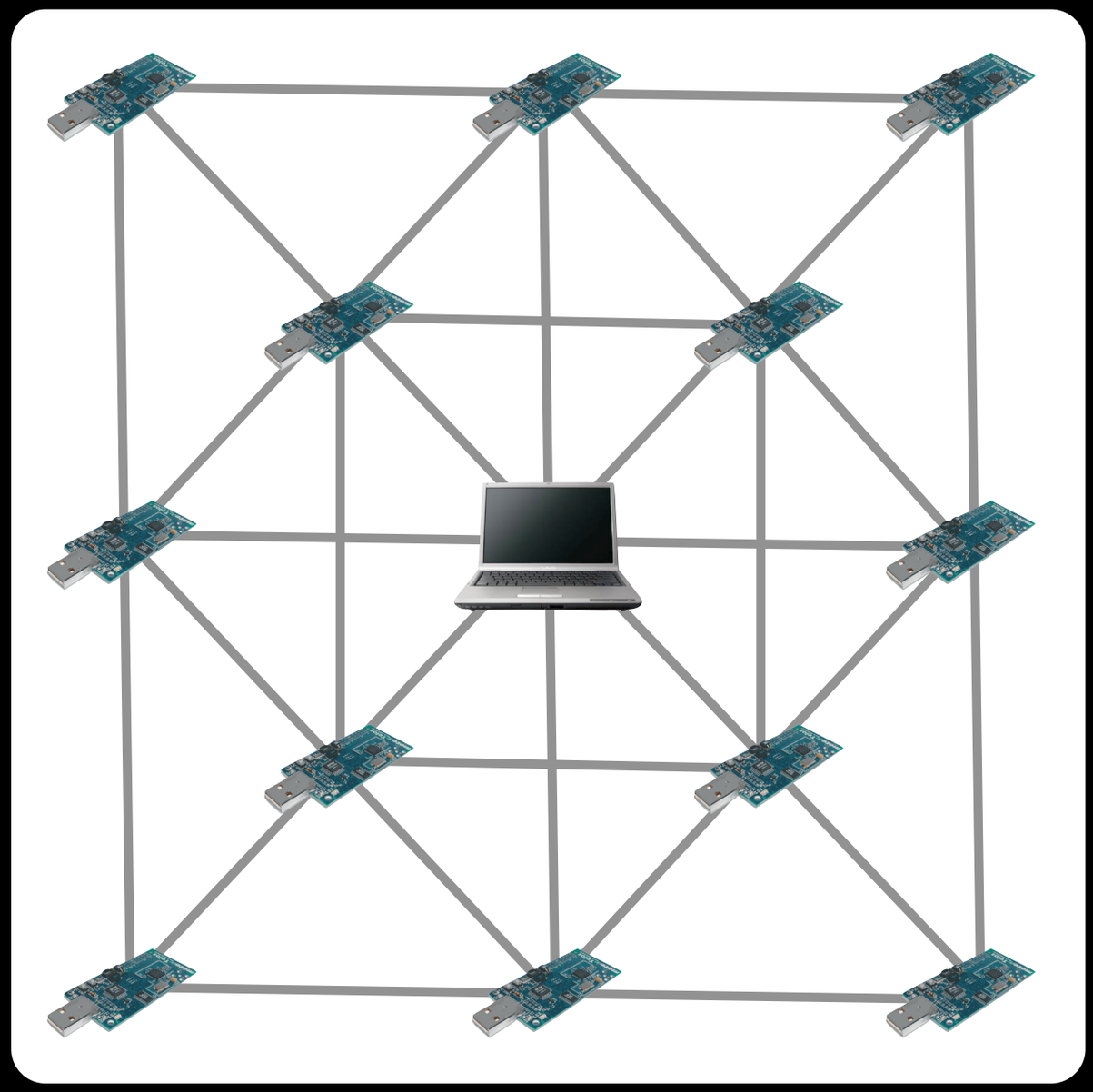
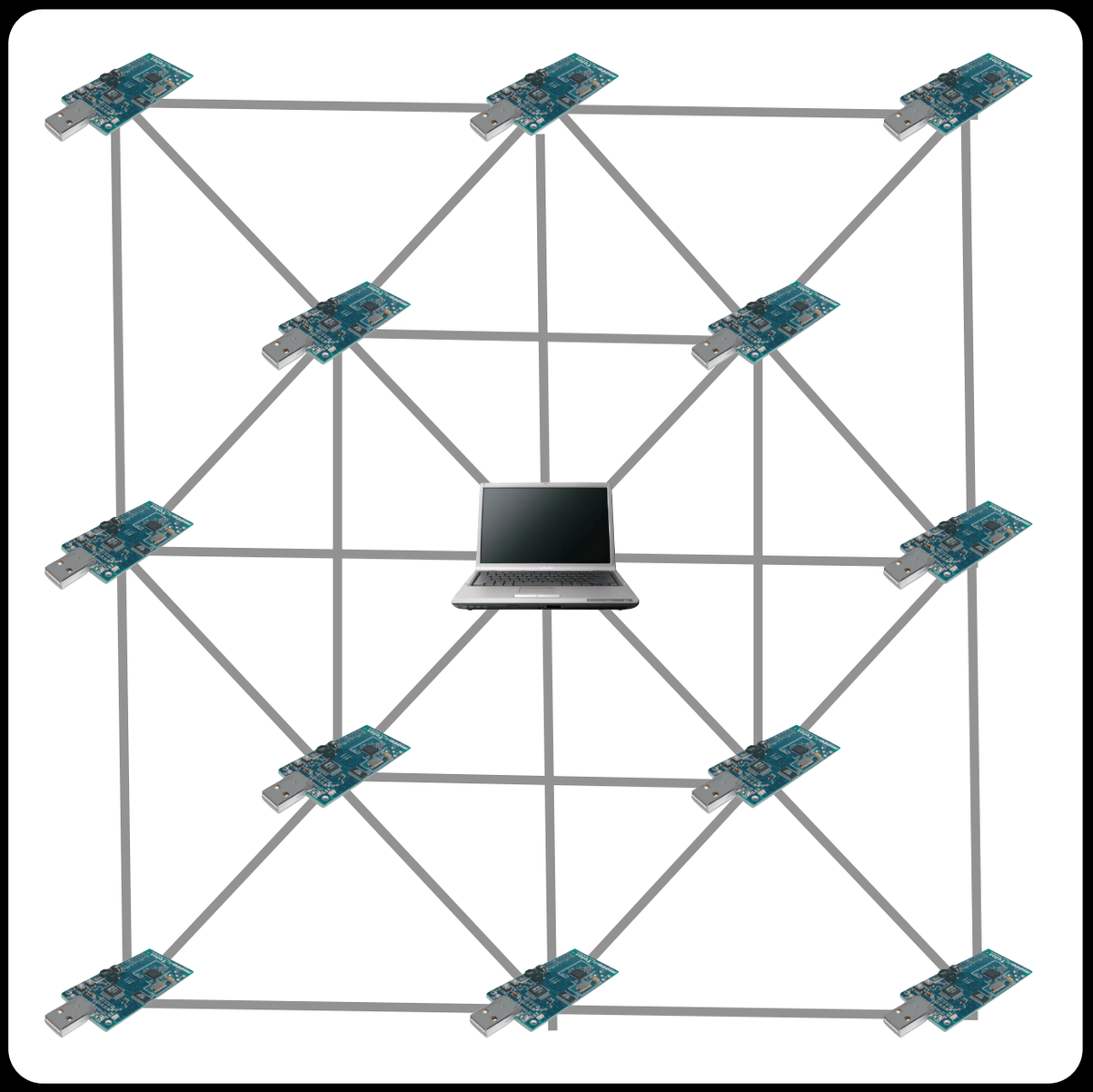
MacroLab: A Vector-based Macroprogramming Framework for Cyber-Physical Systems

Timothy W. Hnat, Tamim I. Sookoor, Pieter Hooimeijer,
Westley Weimer, and Kamin Whitehouse

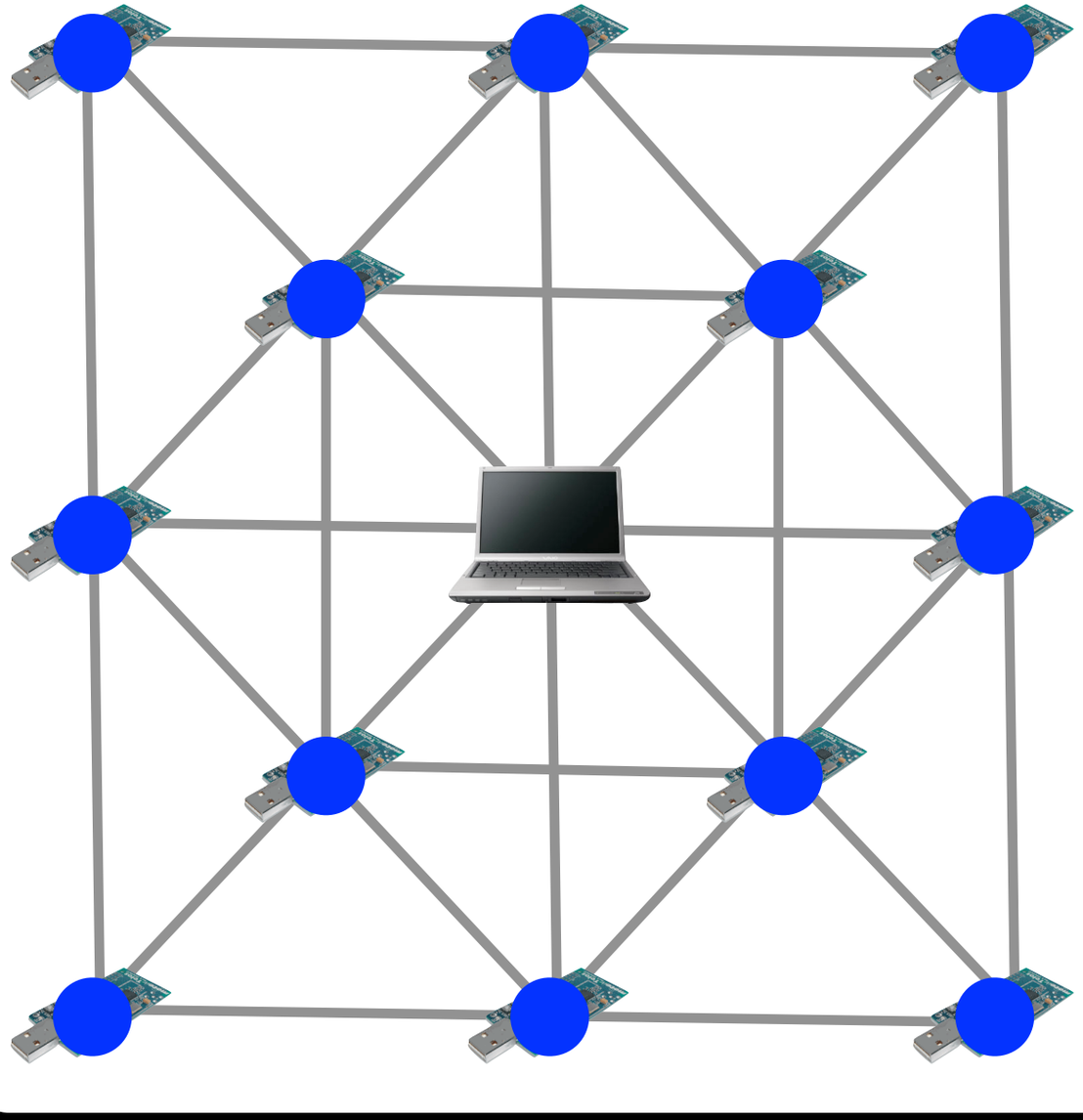
Department of Computer Science
University of Virginia

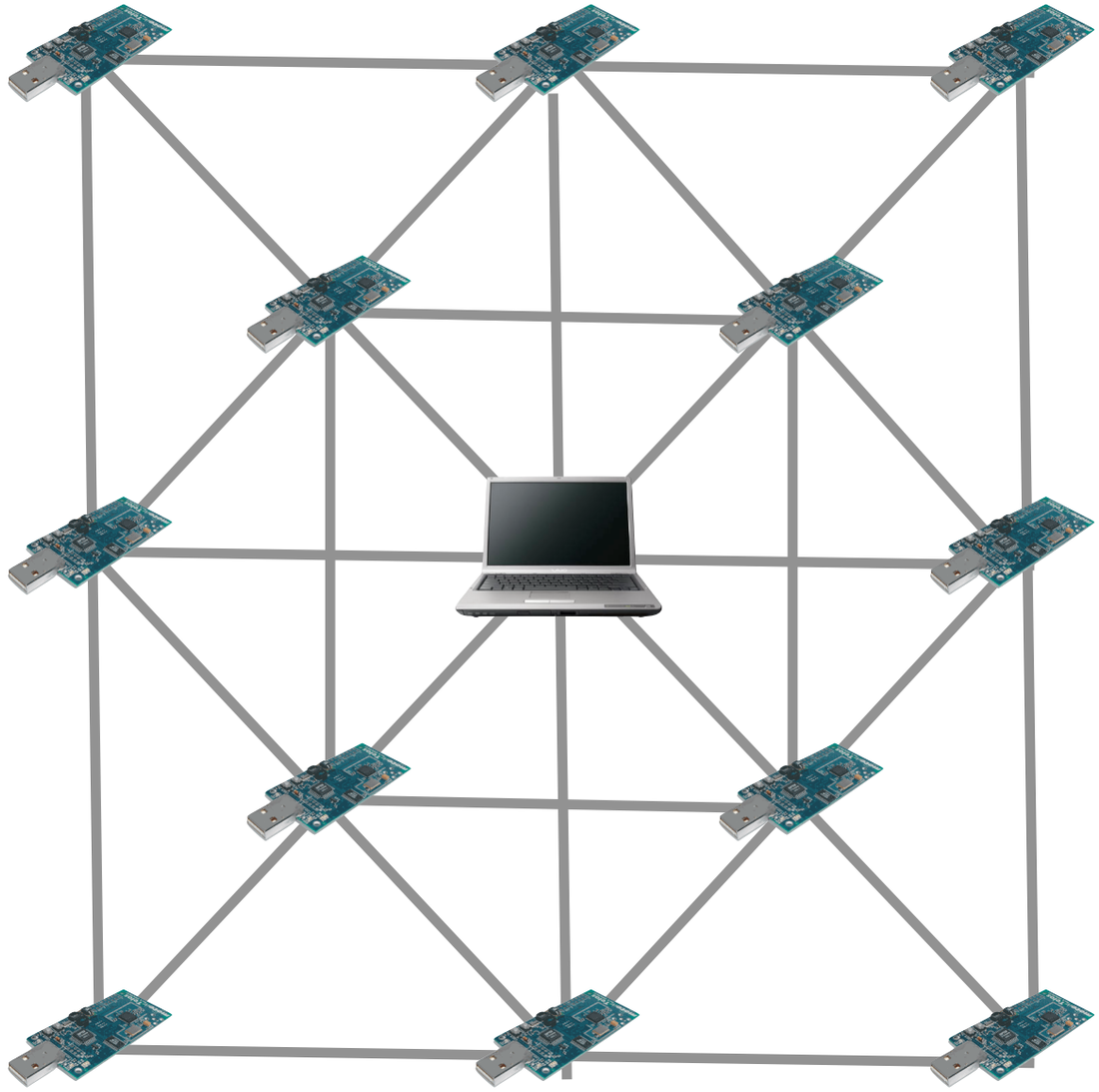
Centralized vs. Distributed



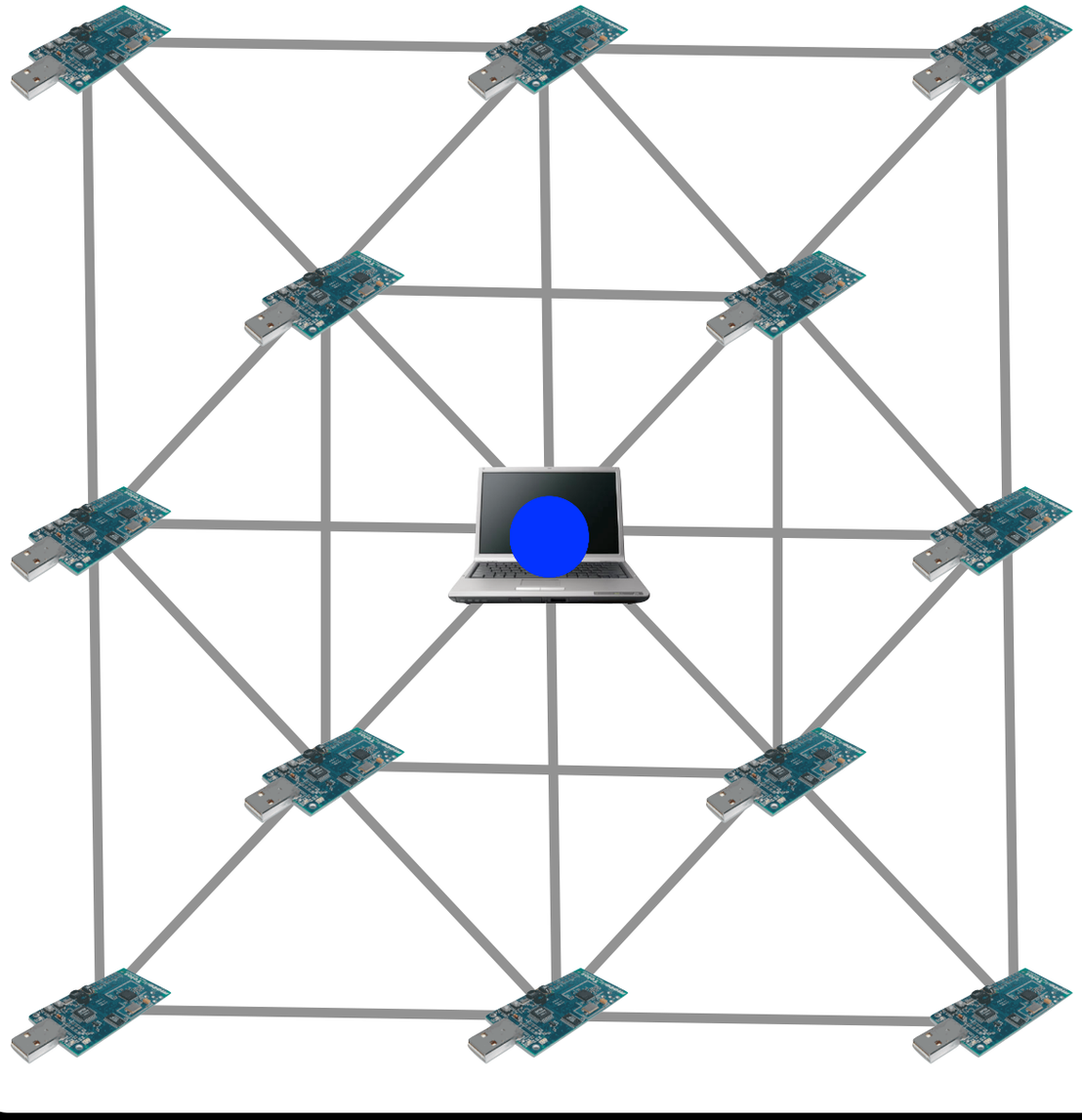


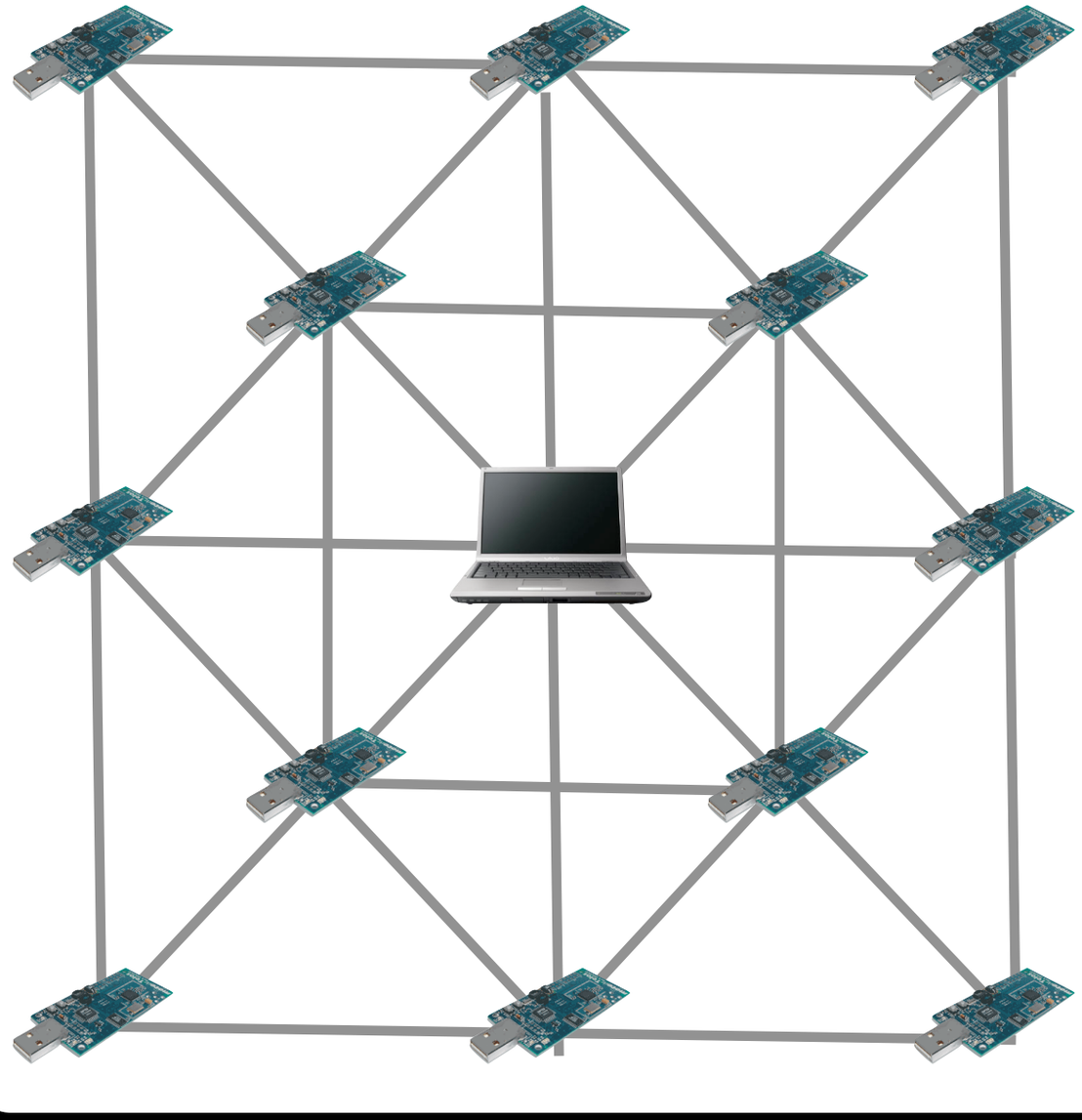
Centralized vs. Distributed



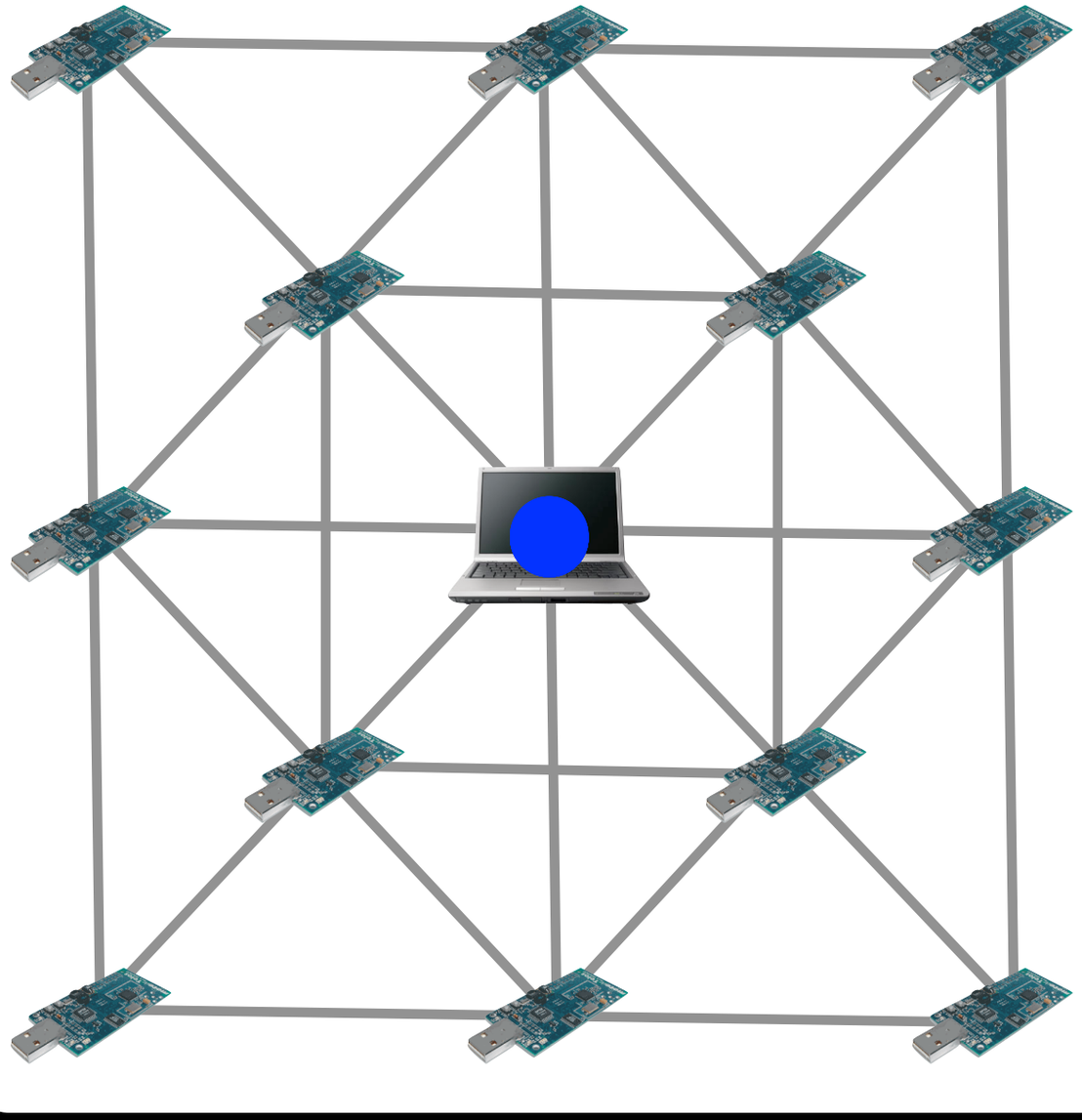


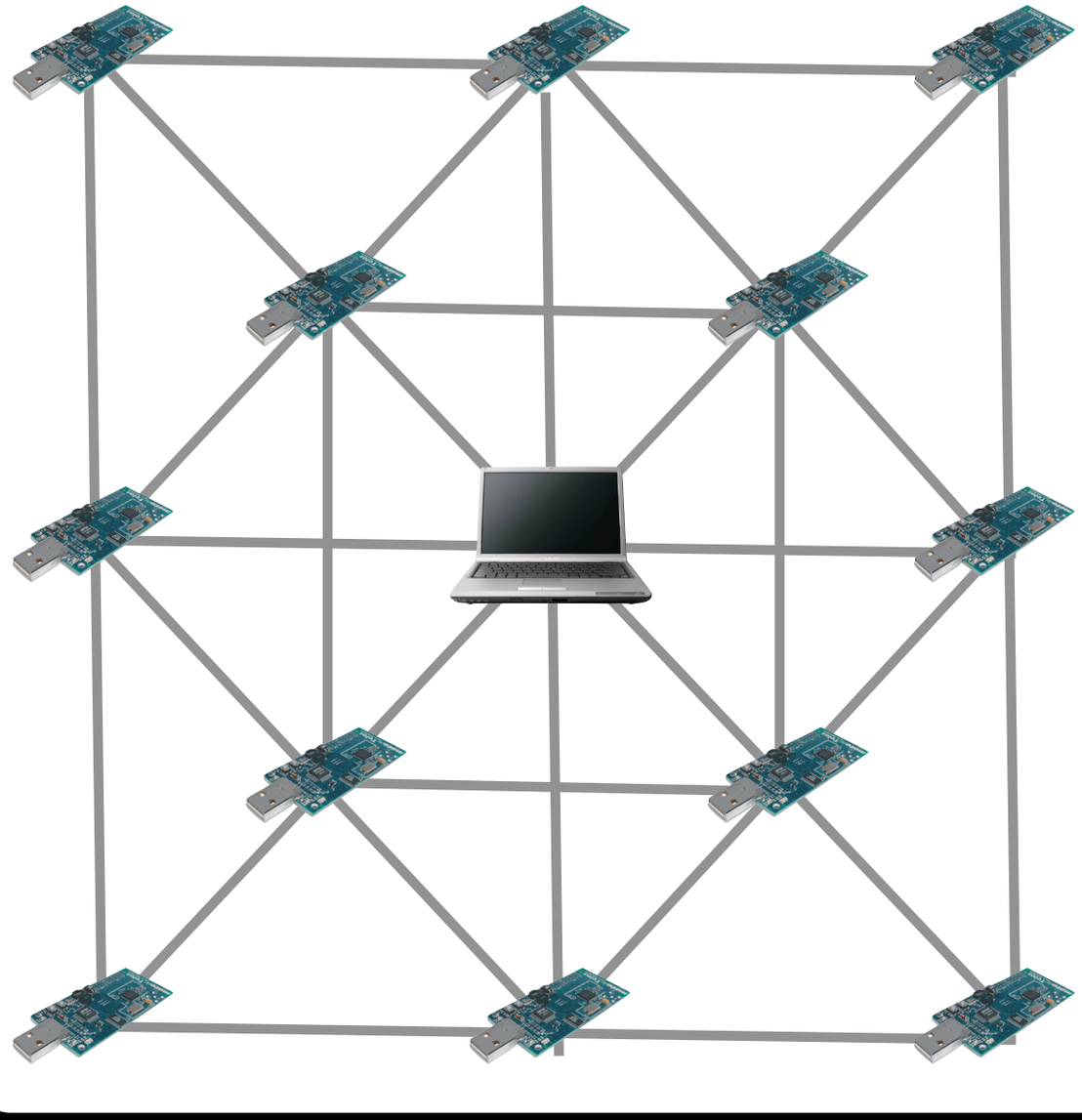
Centralized vs. Distributed



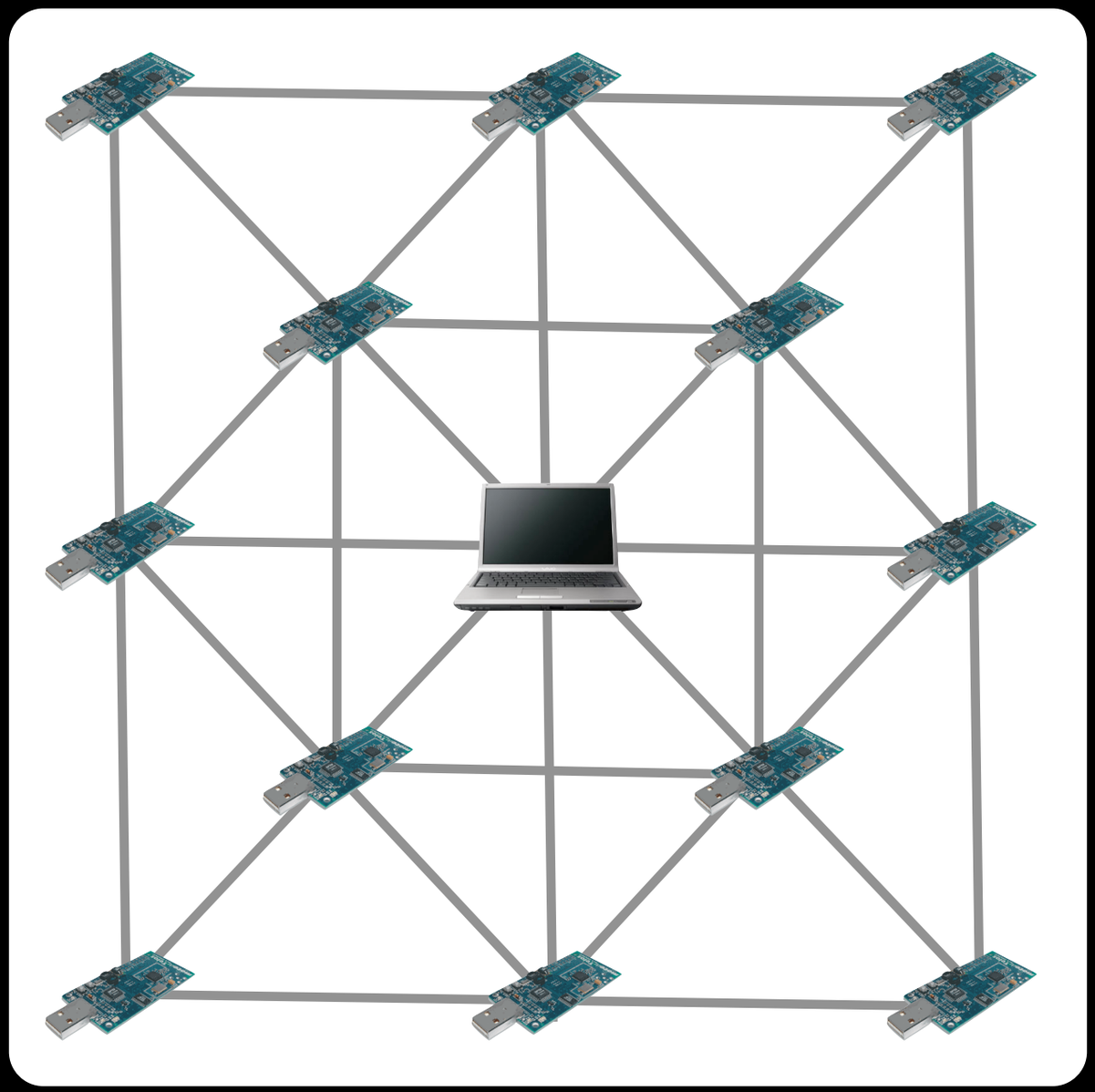


Centralized vs. Distributed

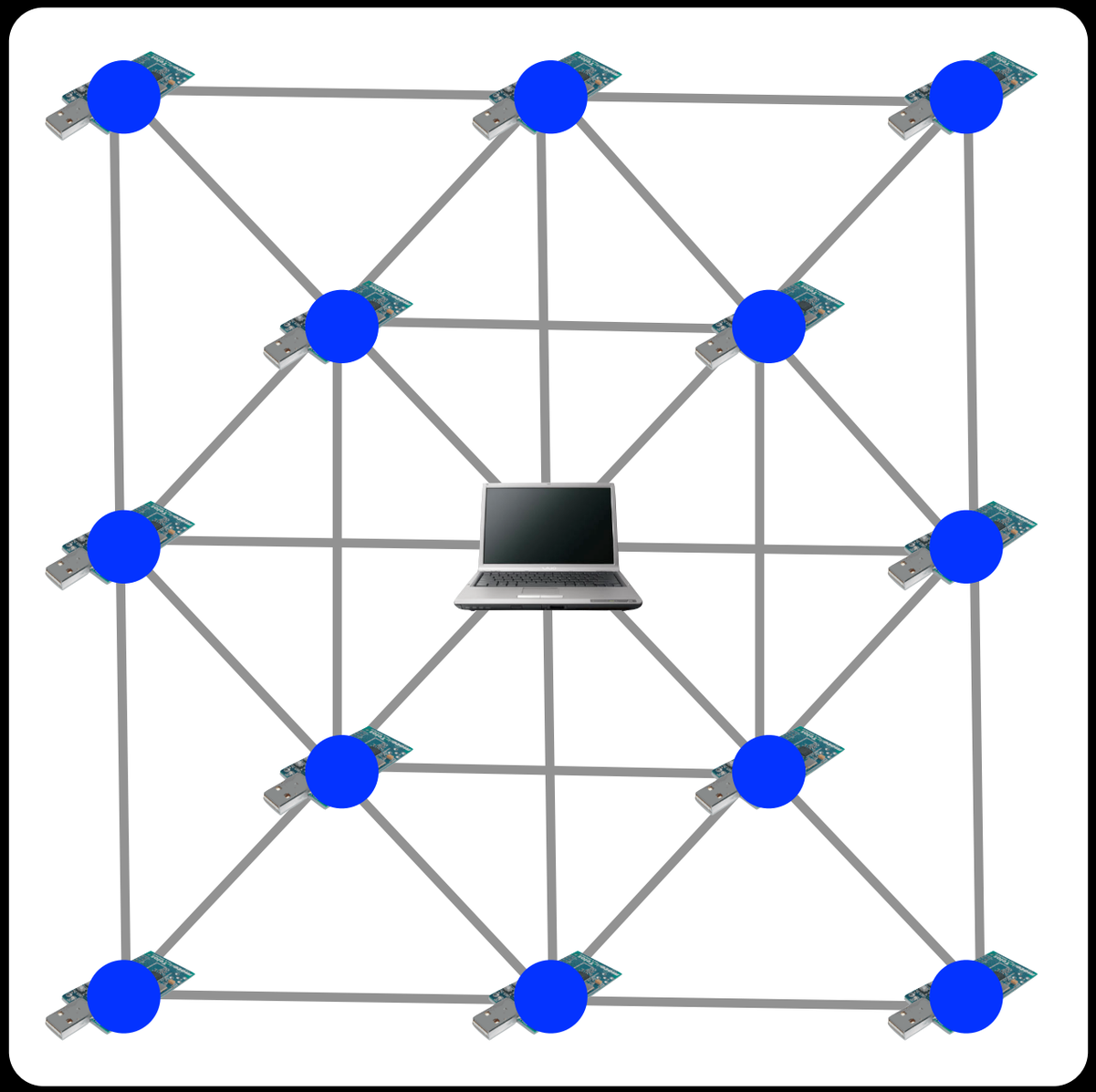




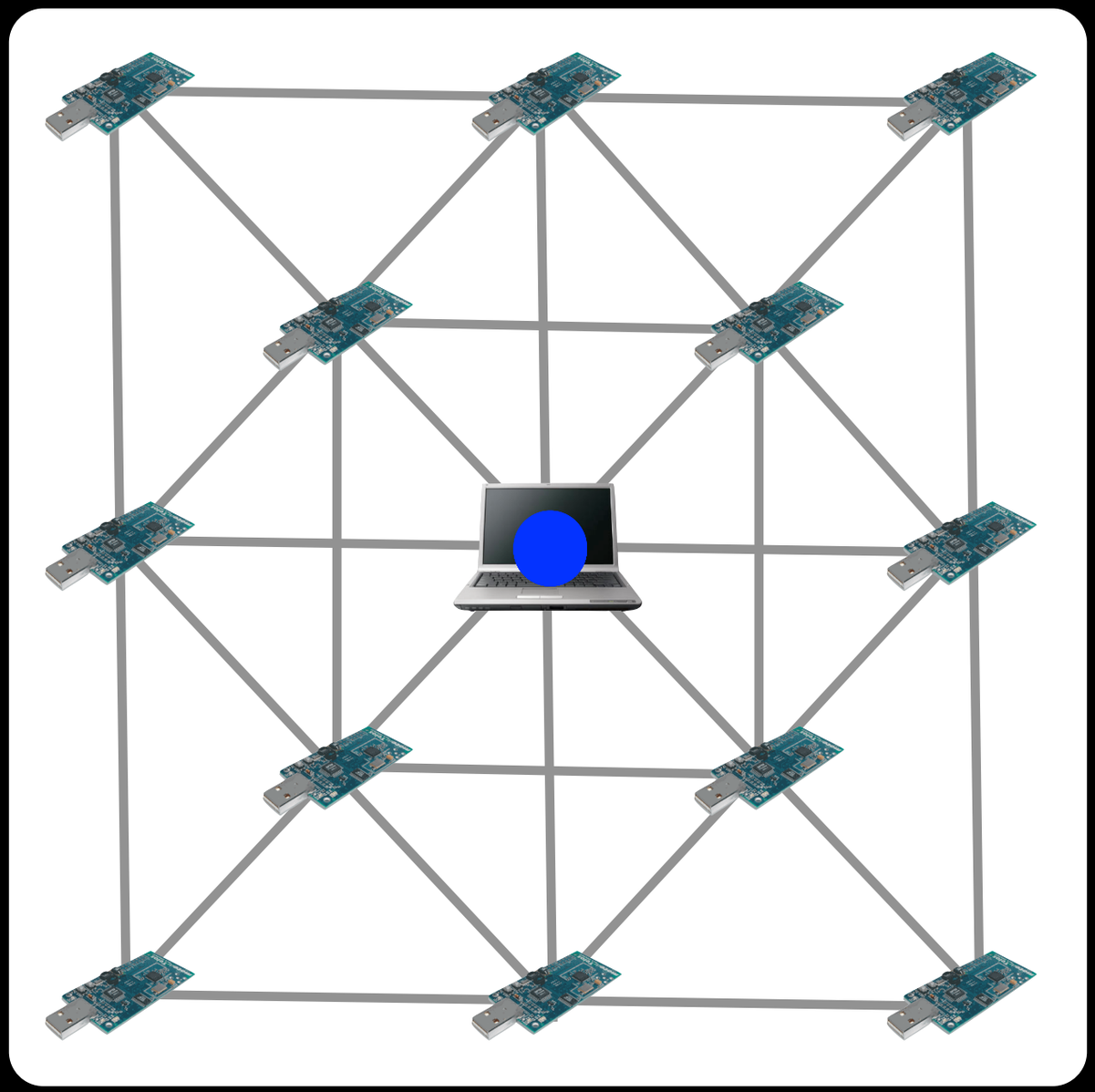
Topology and Logic Matter



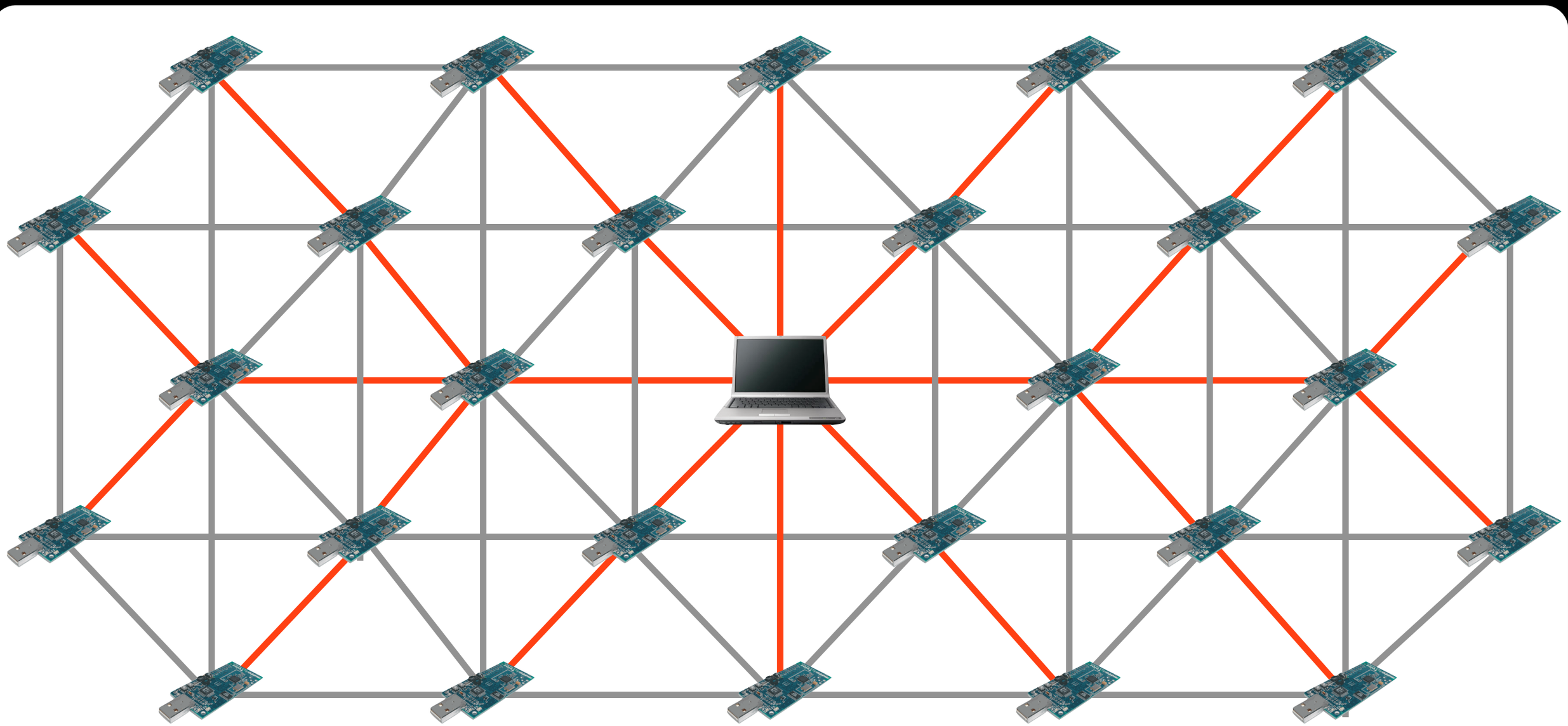
Topology and Logic Matter



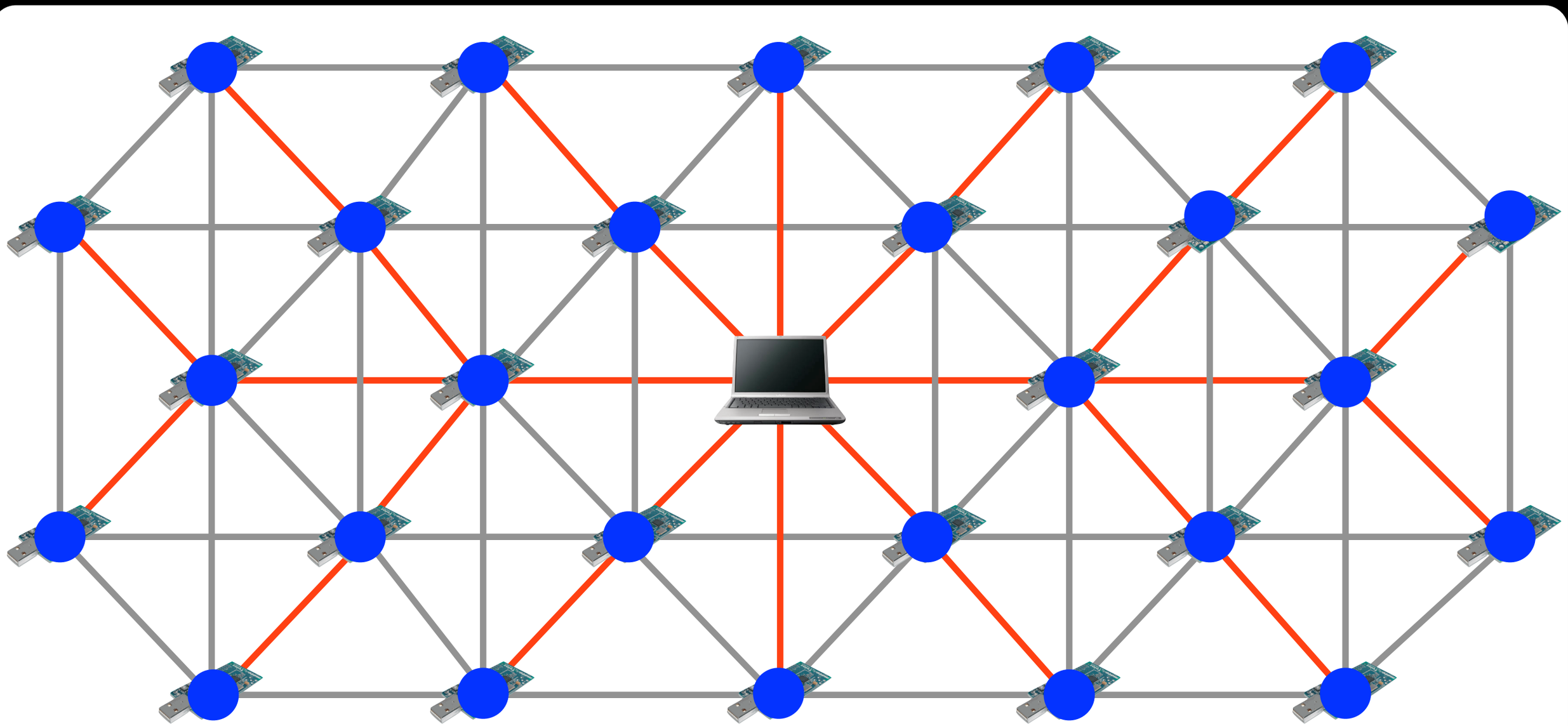
Topology and Logic Matter



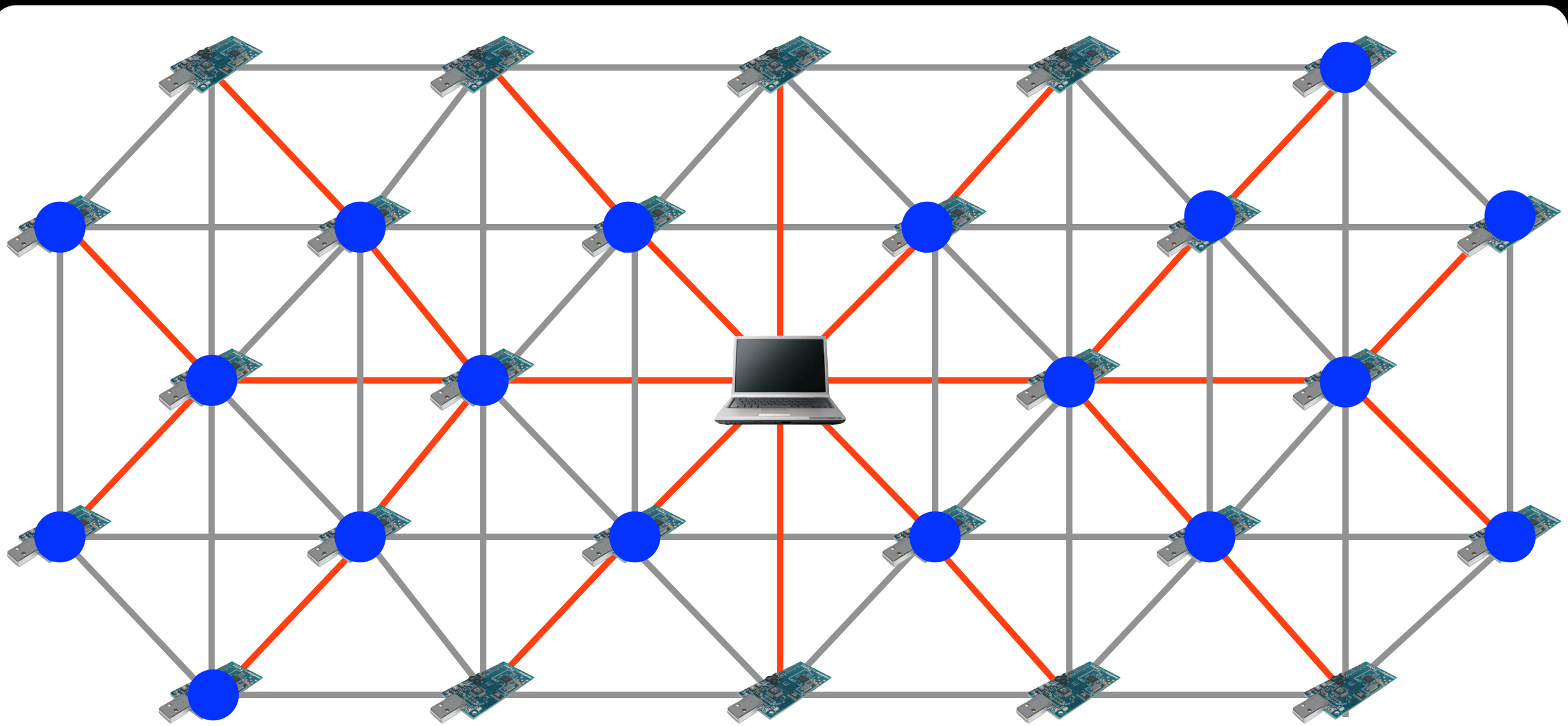
Topology and Logic Matter



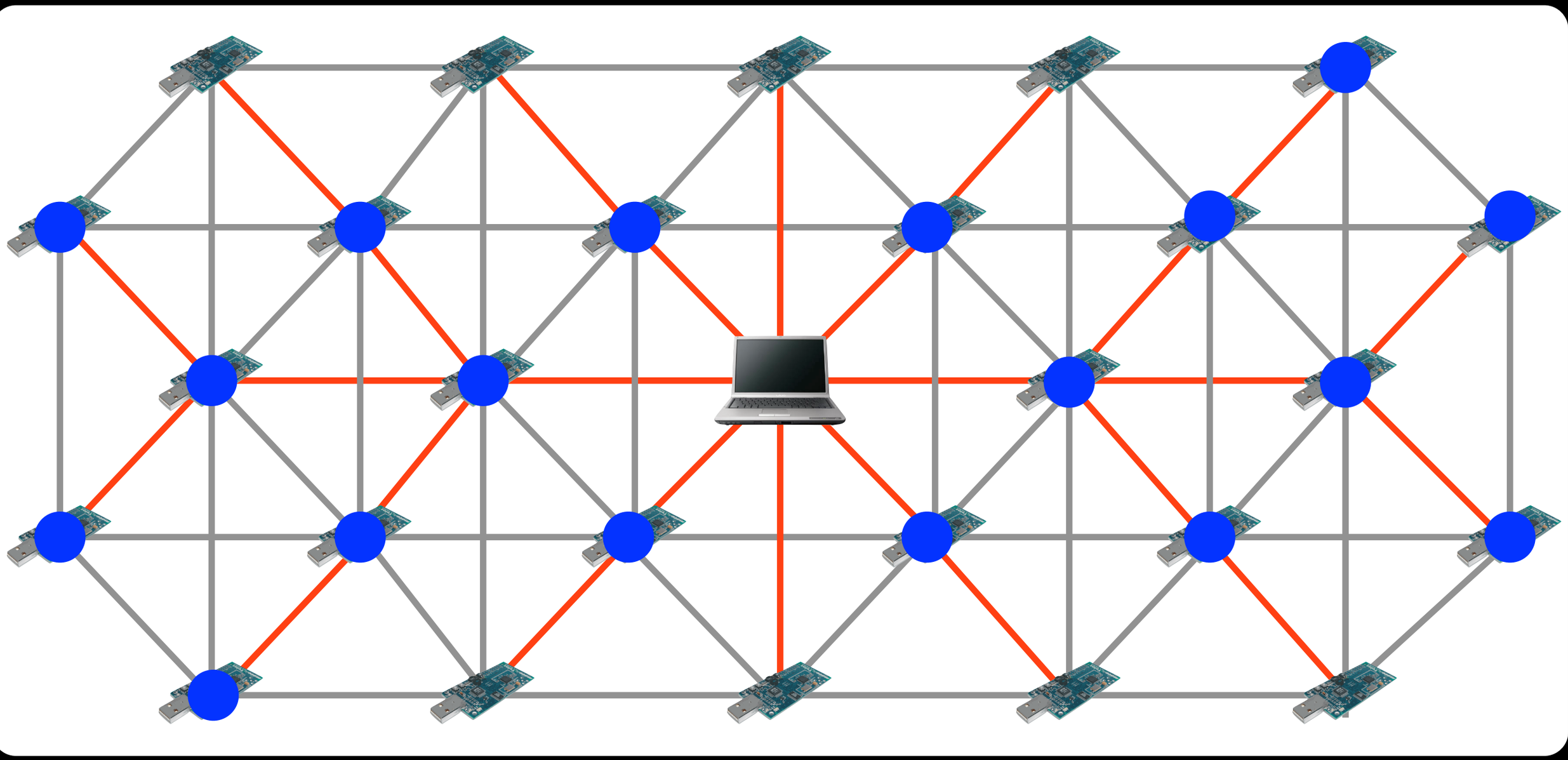
Topology and Logic Matter



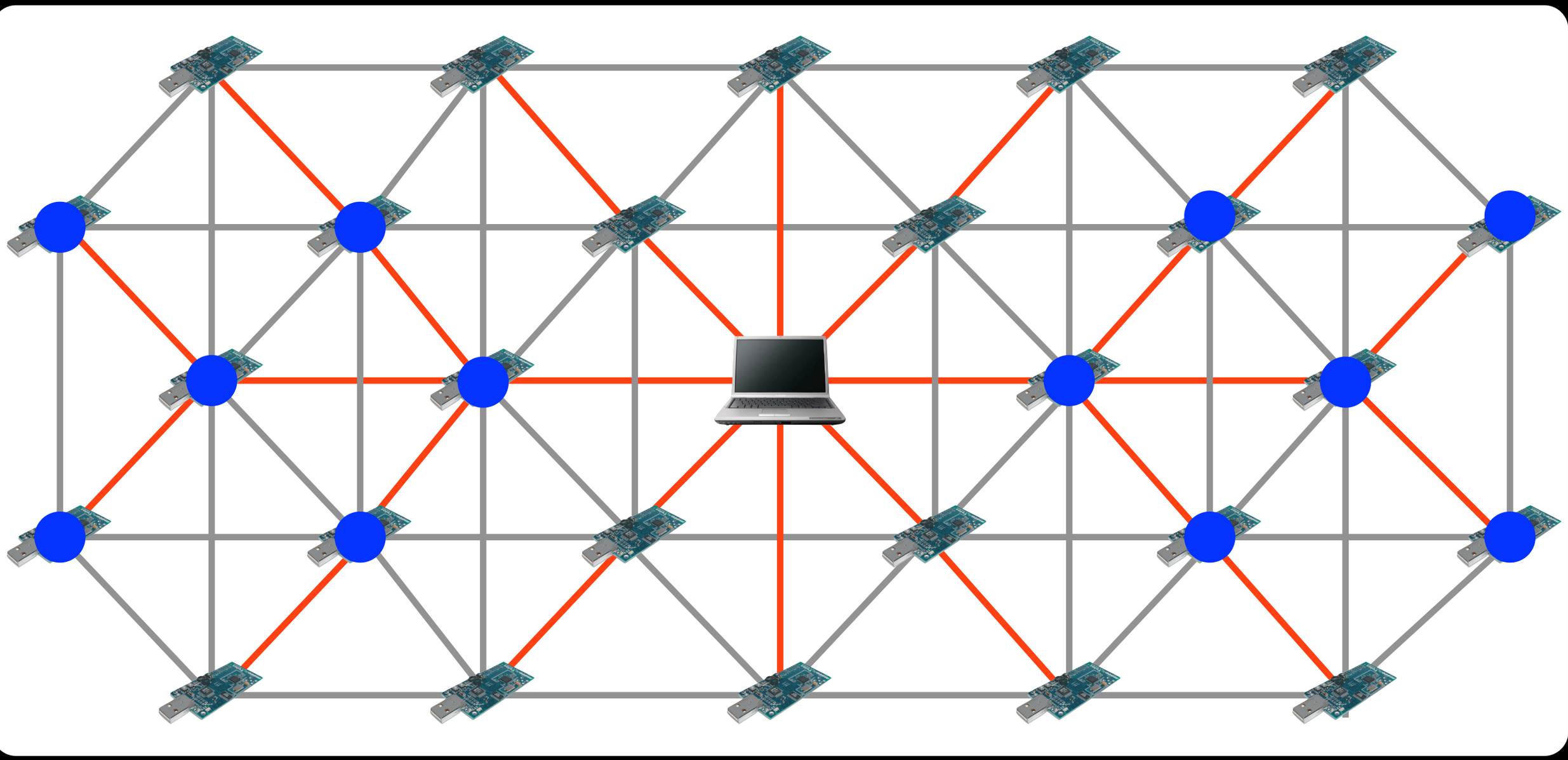
Topology and Logic Matter



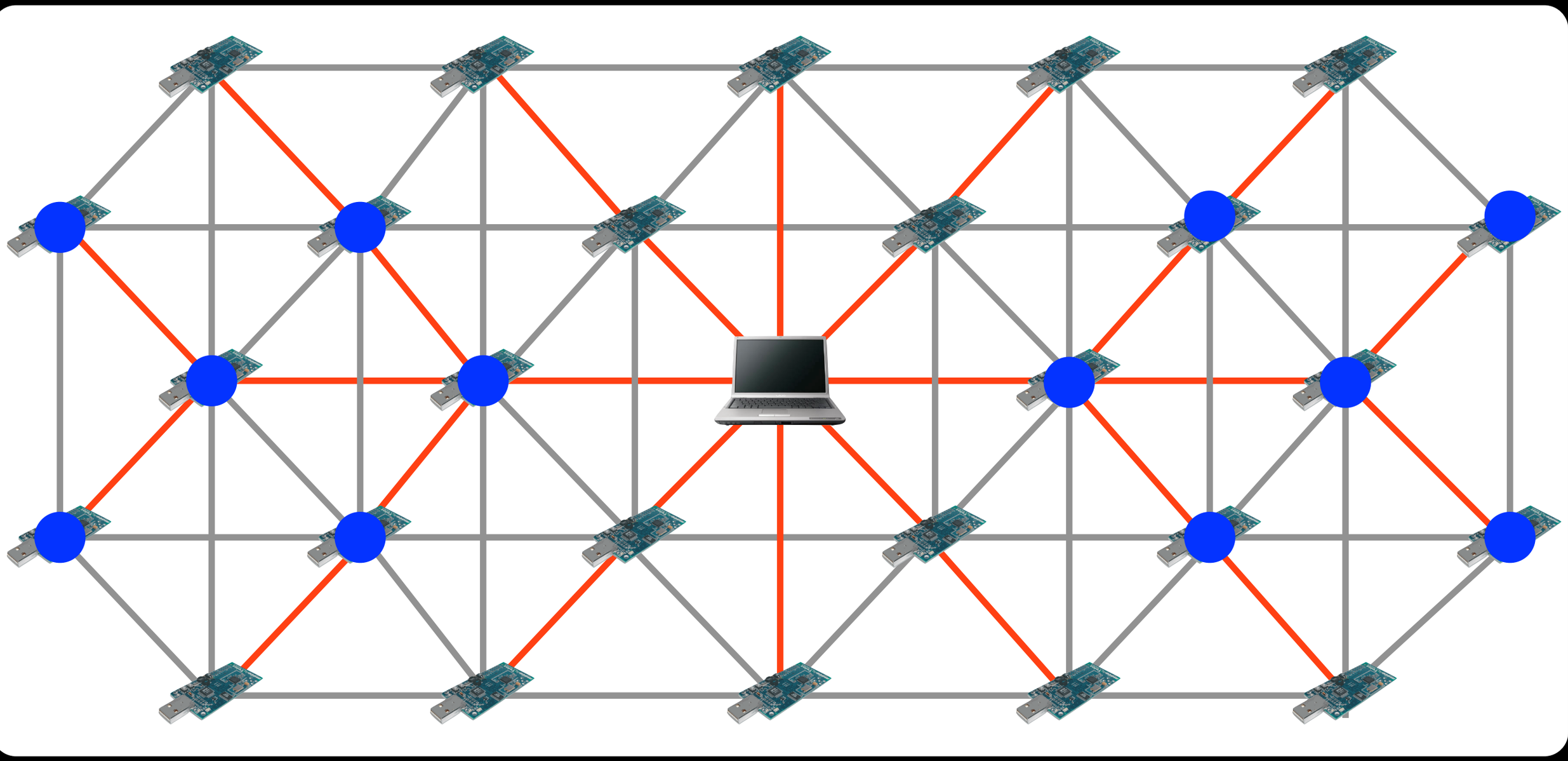
Topology and Logic Matter



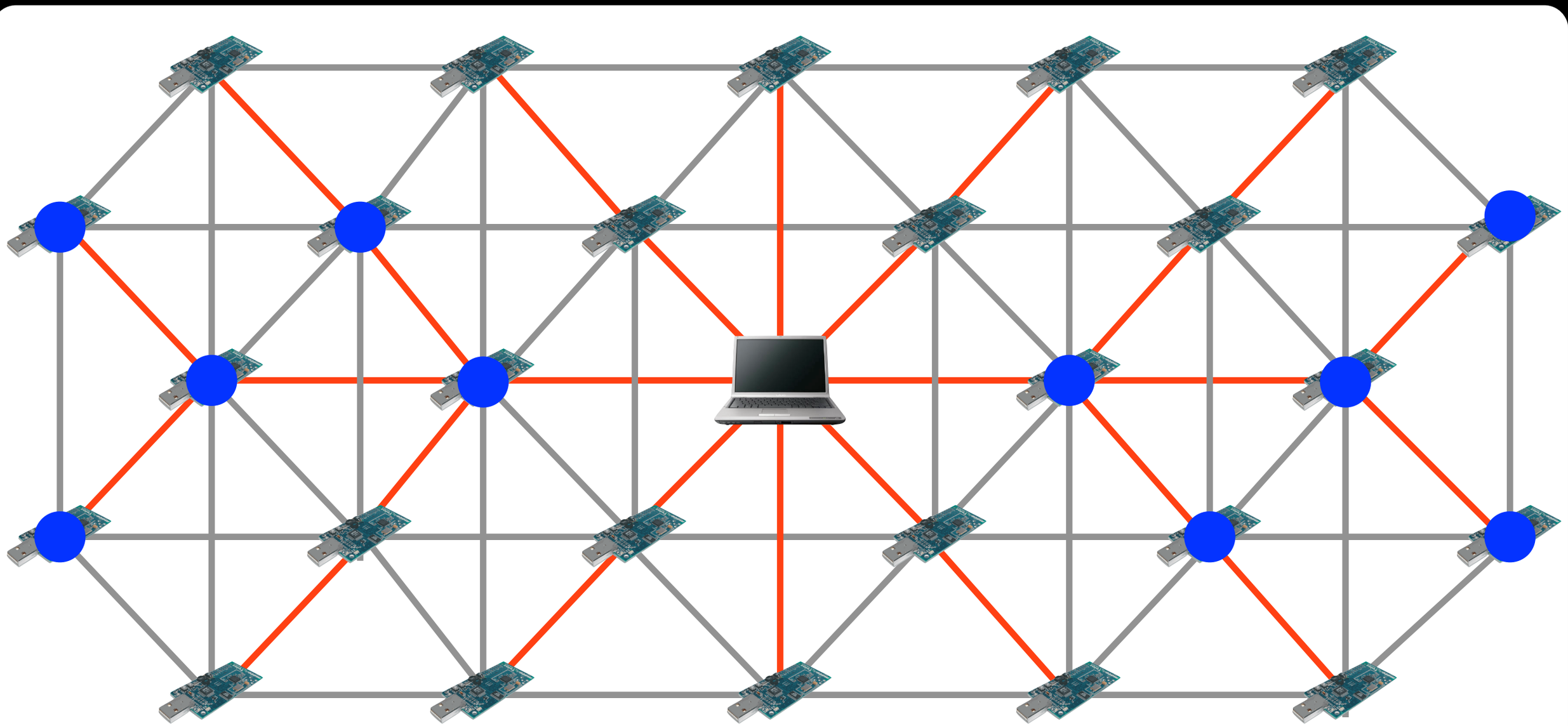
Topology and Logic Matter



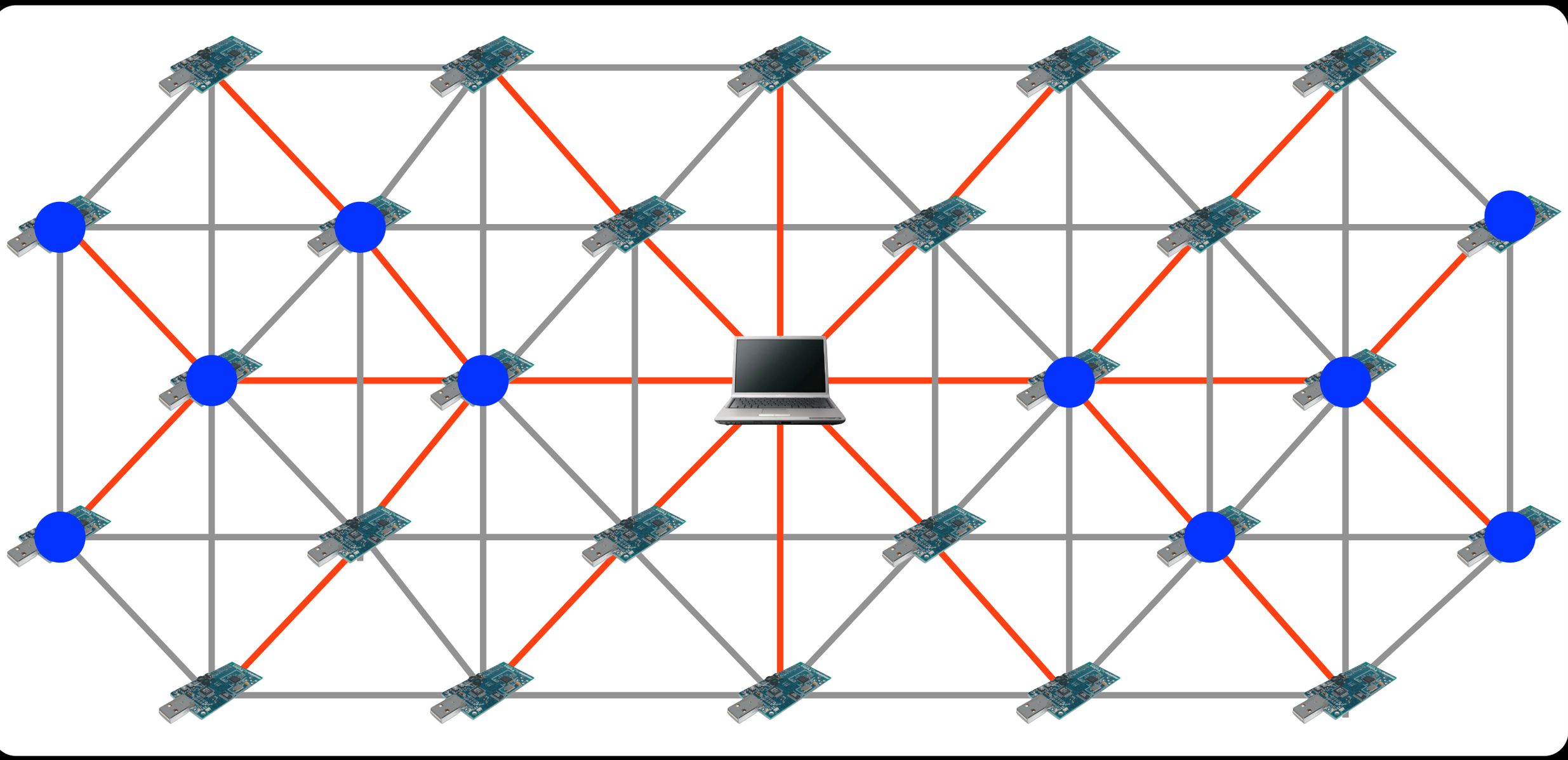
Topology and Logic Matter



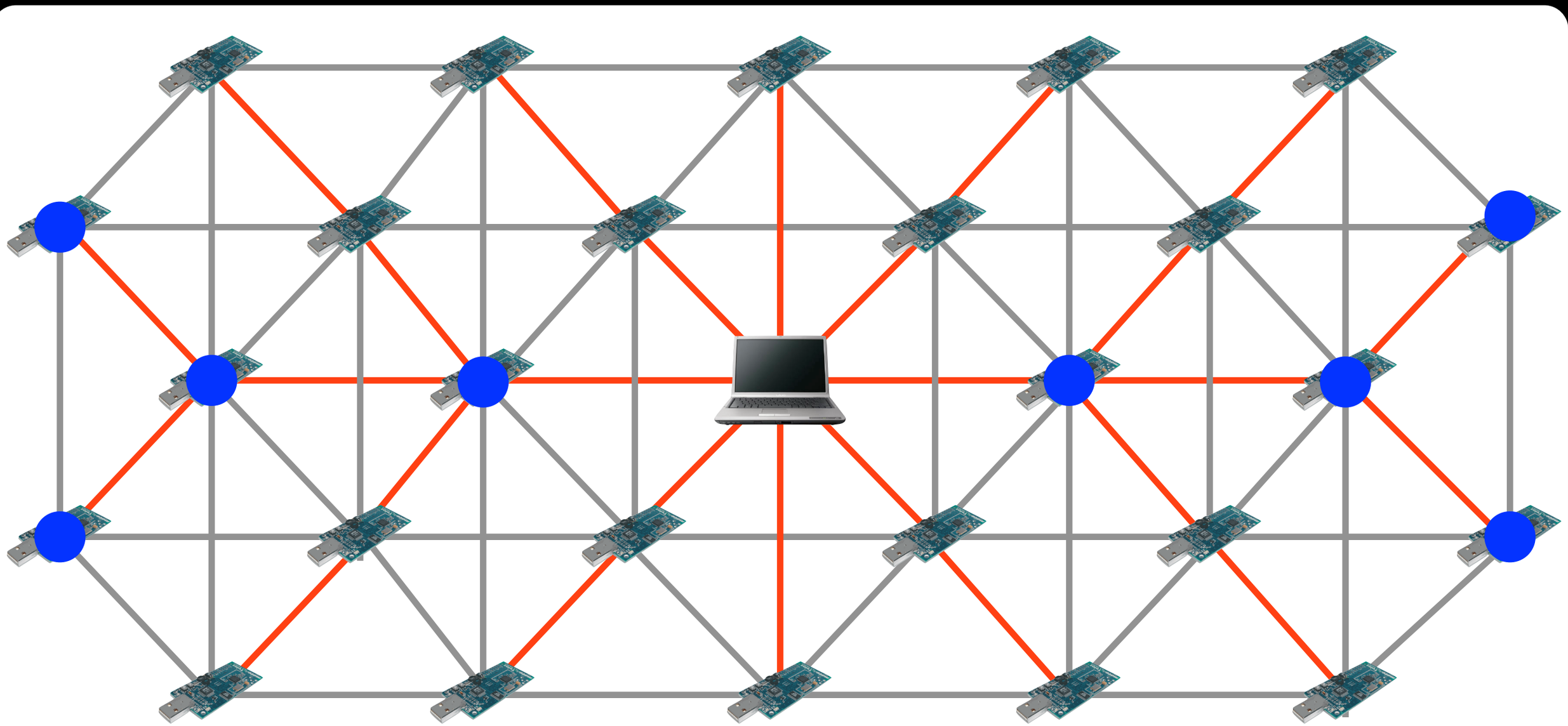
Topology and Logic Matter



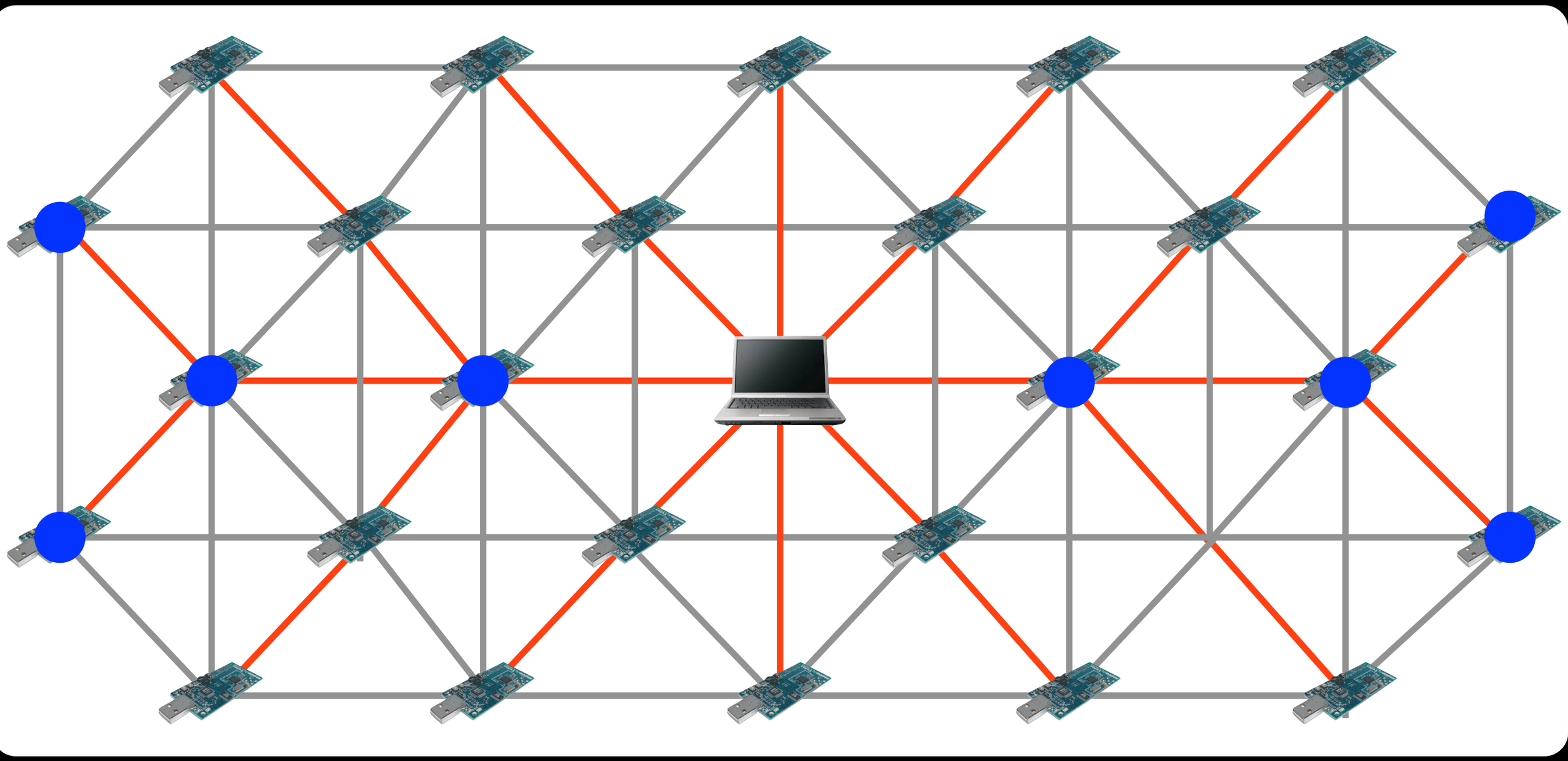
Topology and Logic Matter



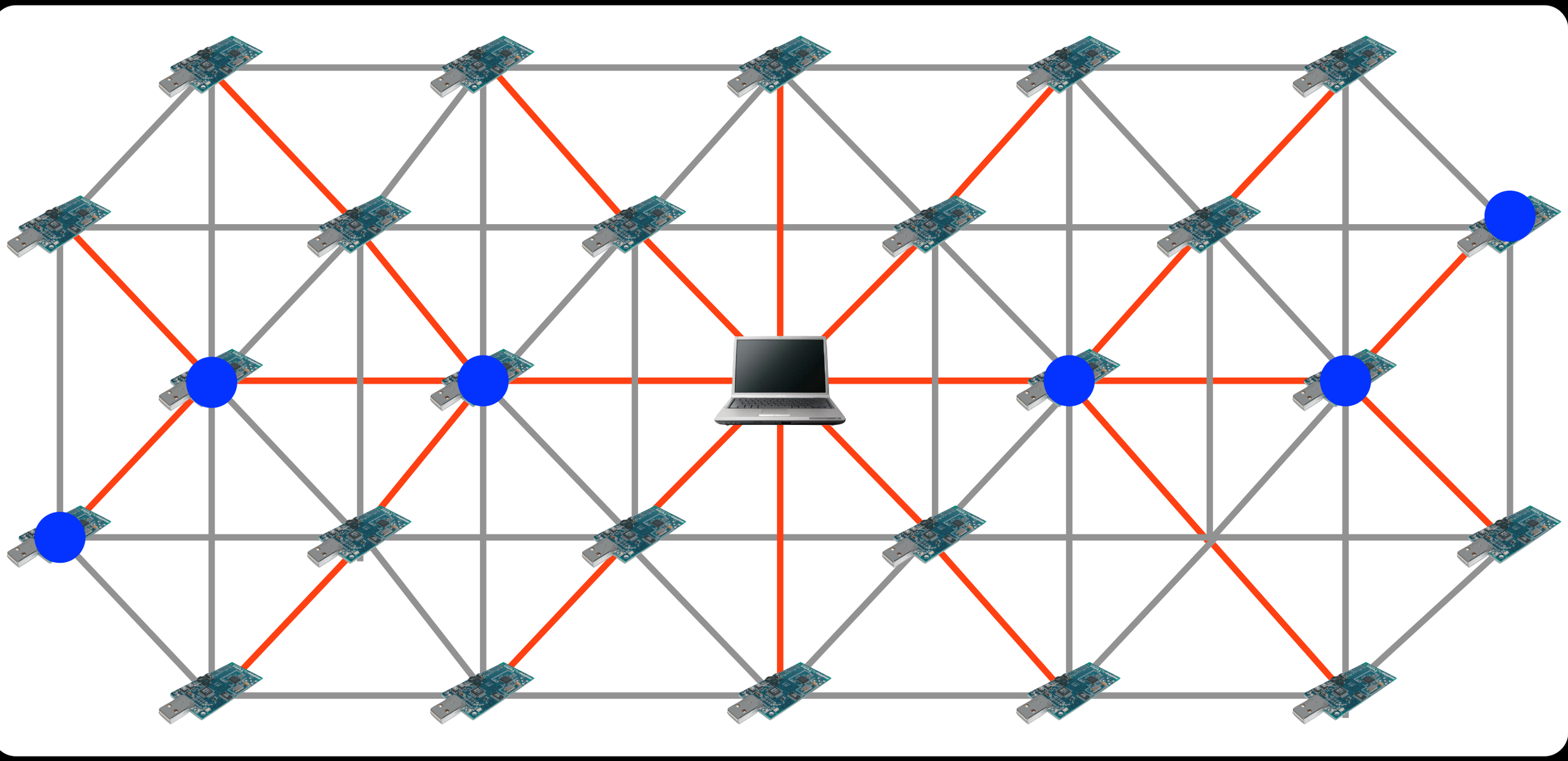
Topology and Logic Matter



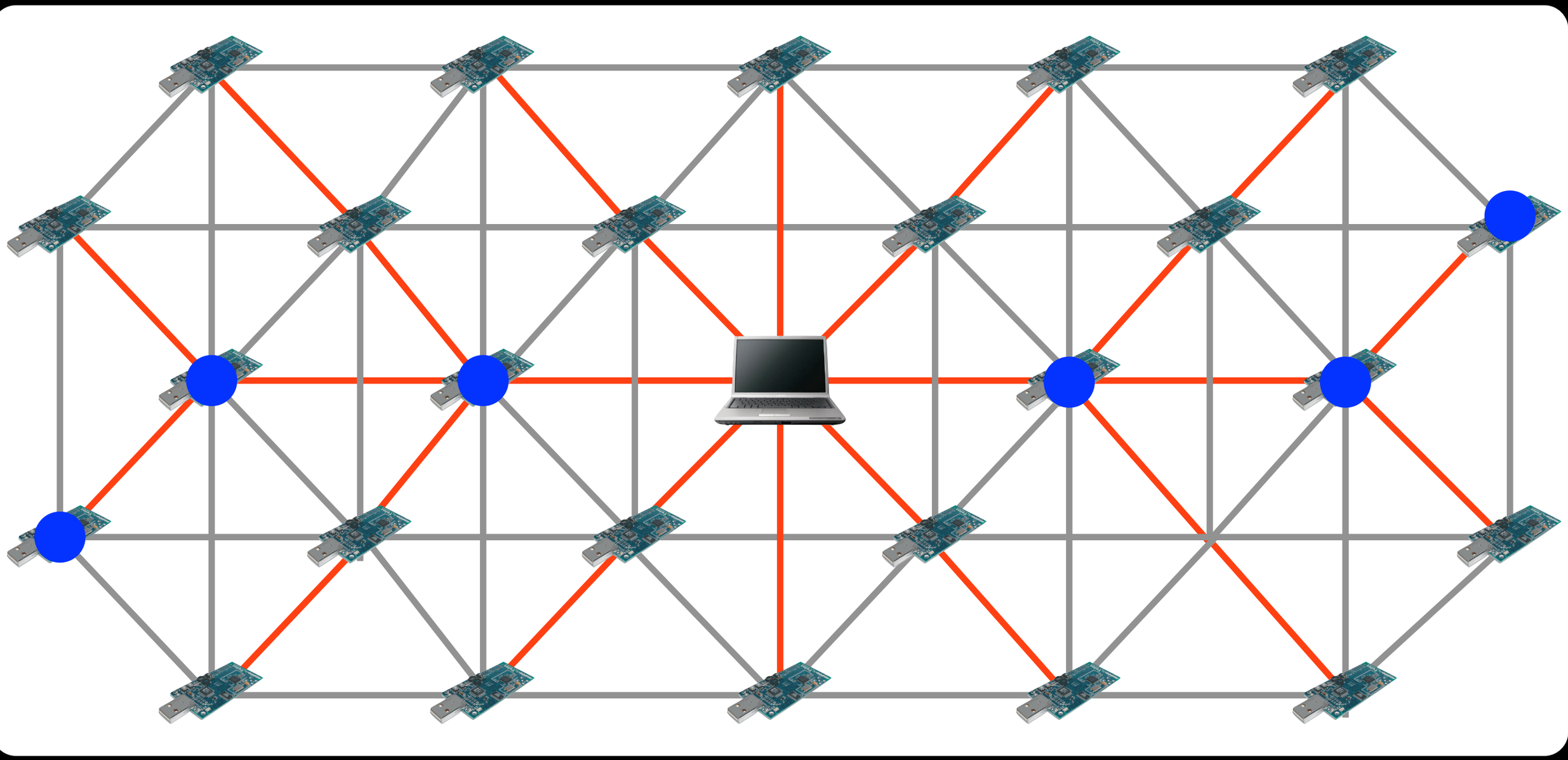
Topology and Logic Matter



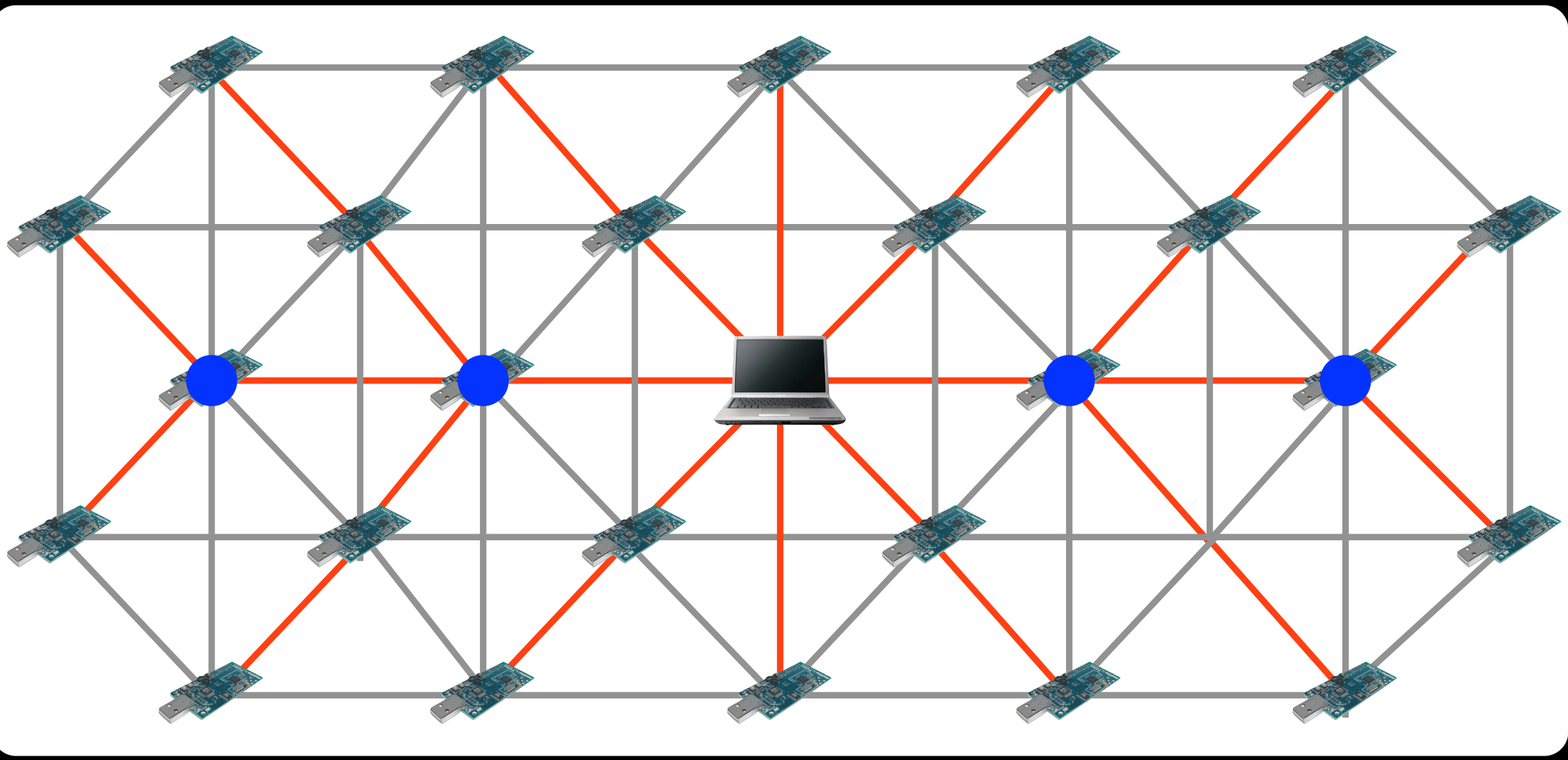
Topology and Logic Matter



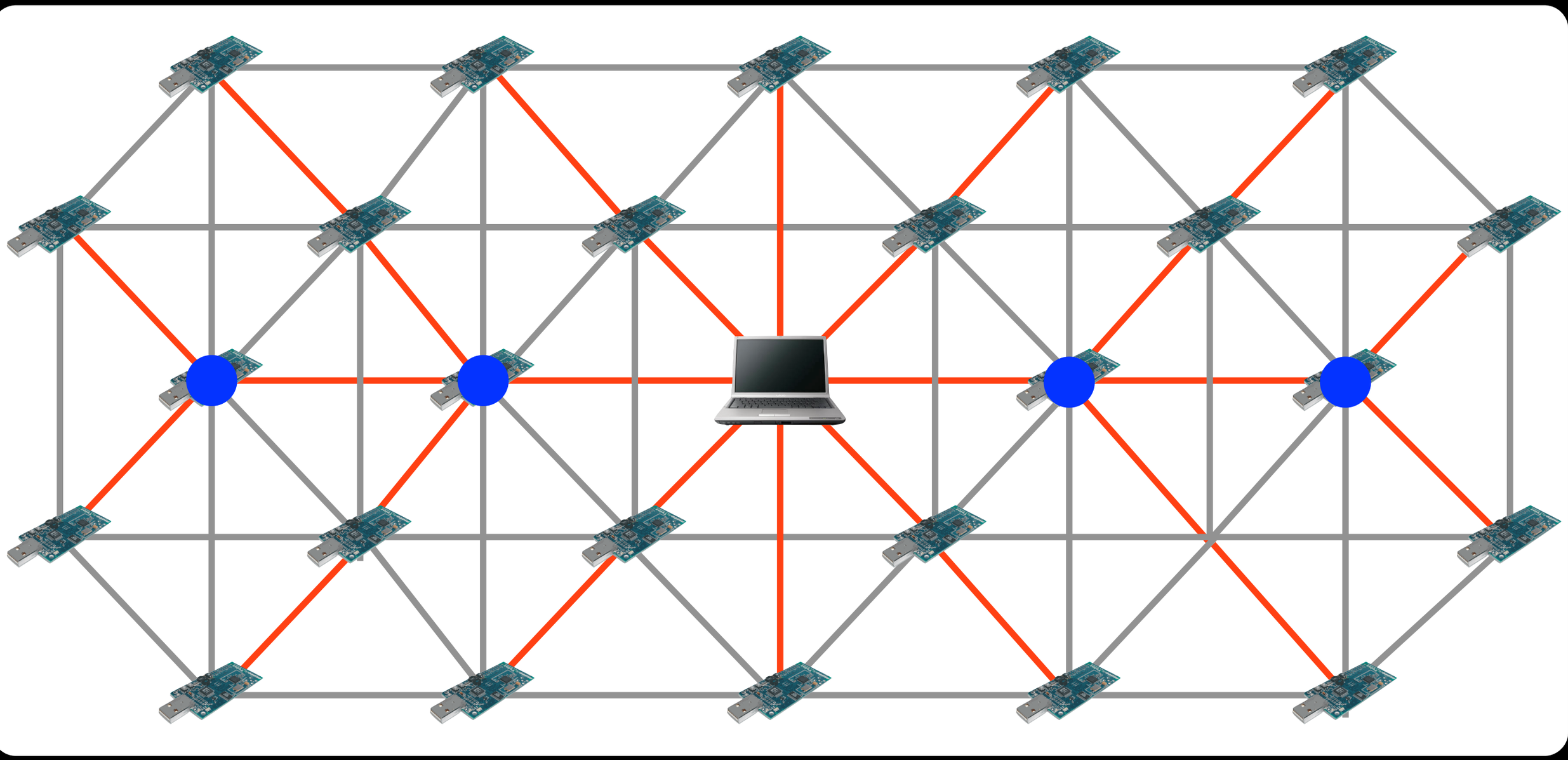
Topology and Logic Matter



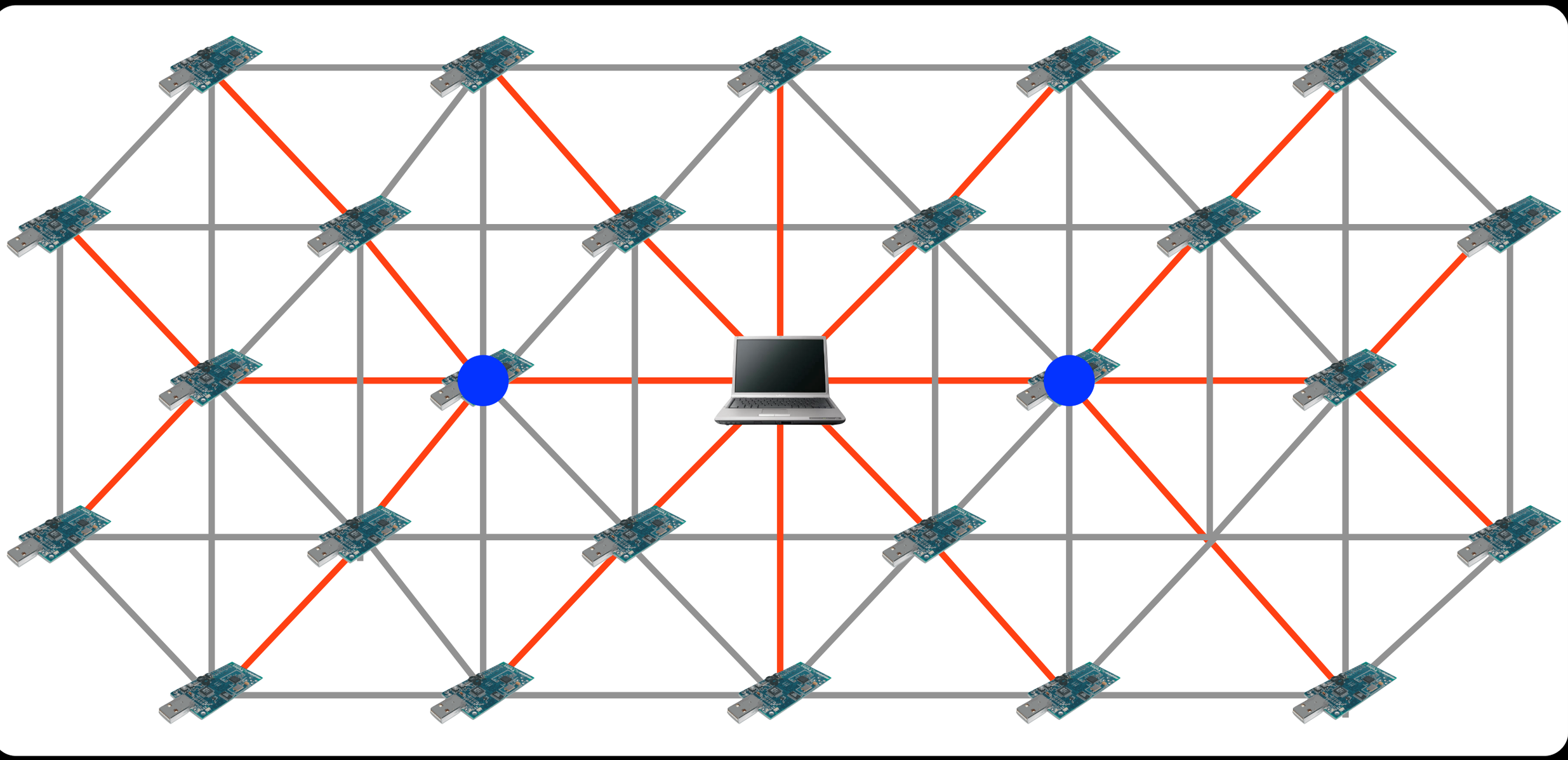
Topology and Logic Matter



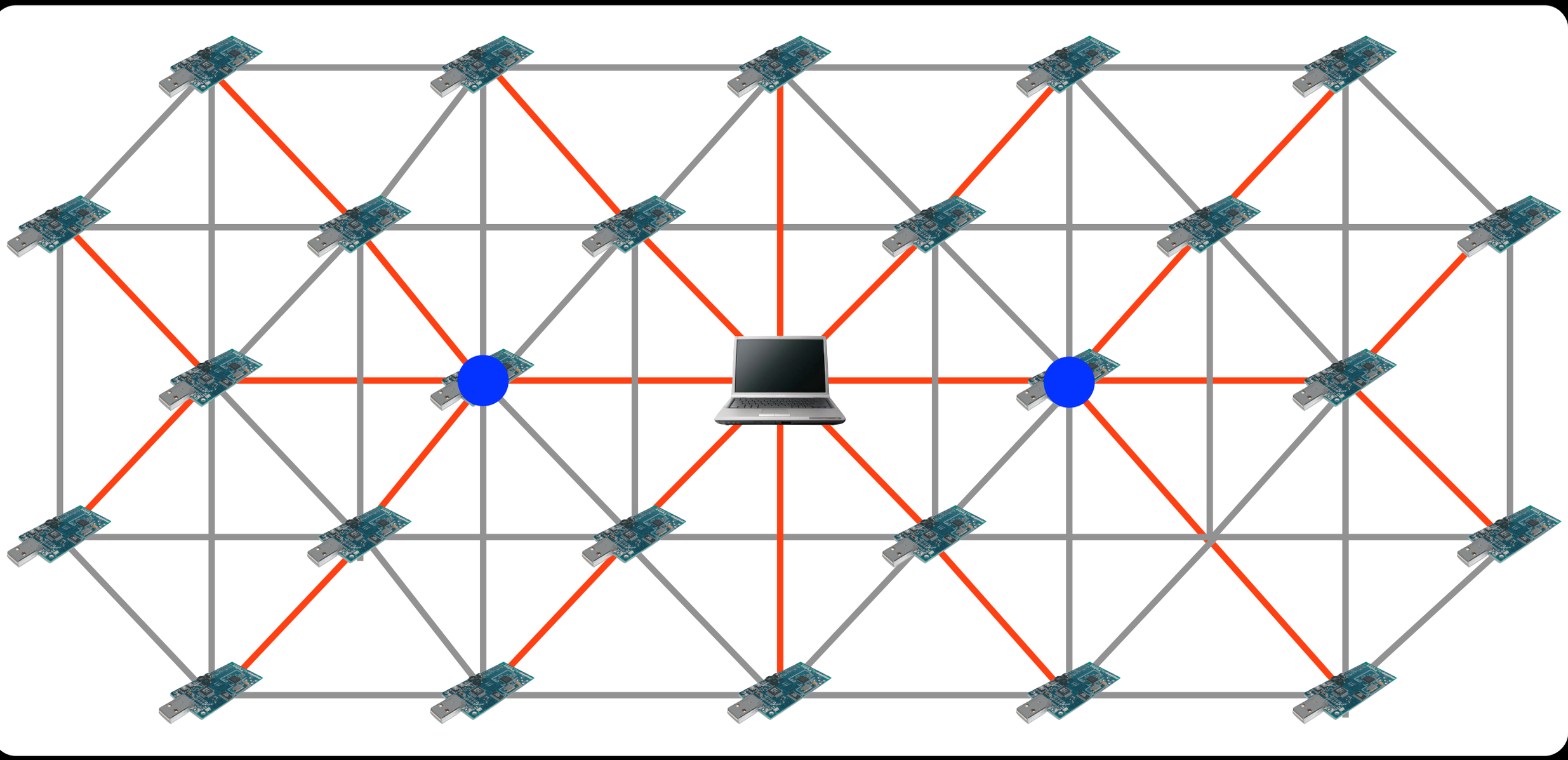
Topology and Logic Matter



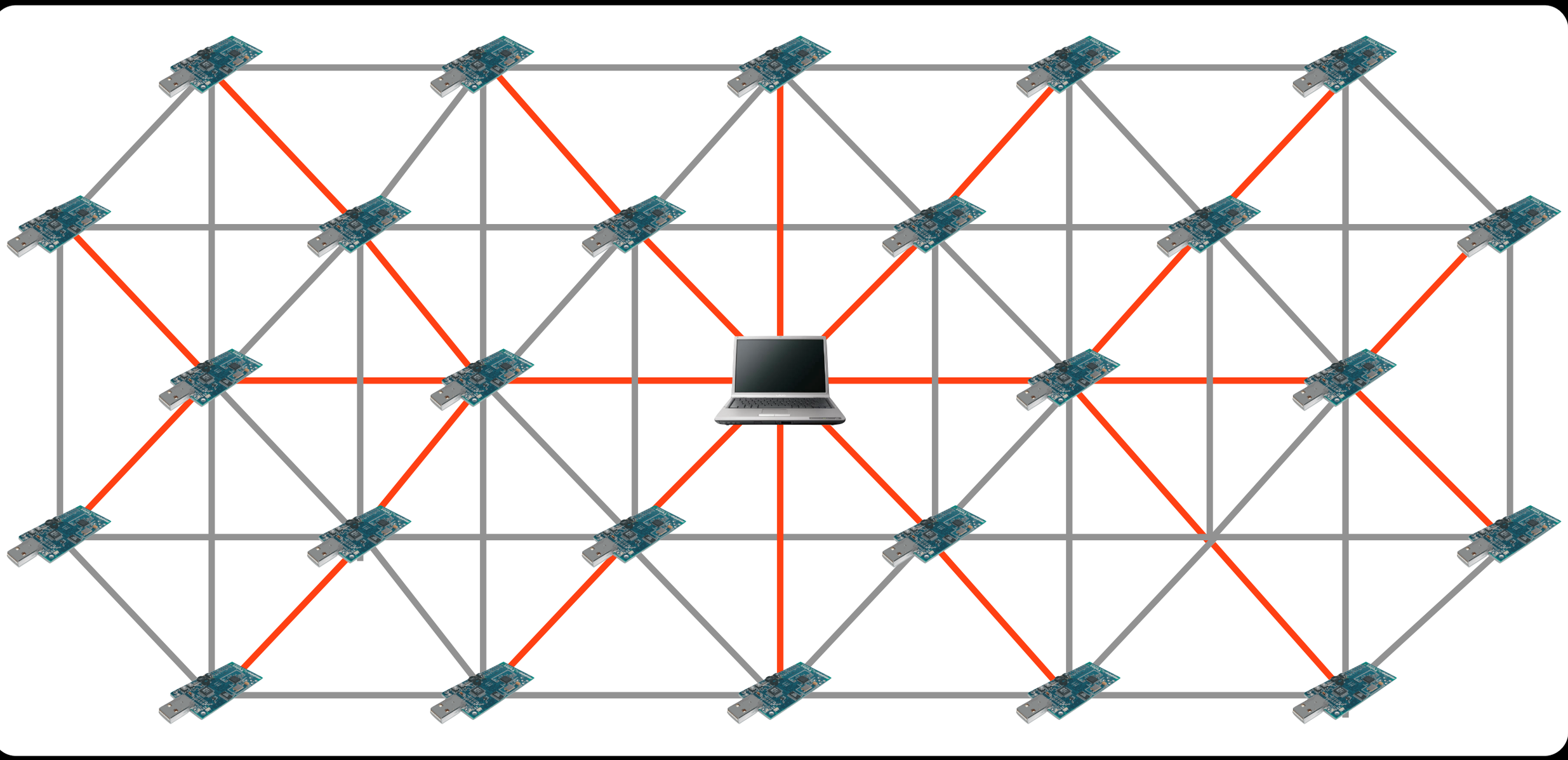
Topology and Logic Matter



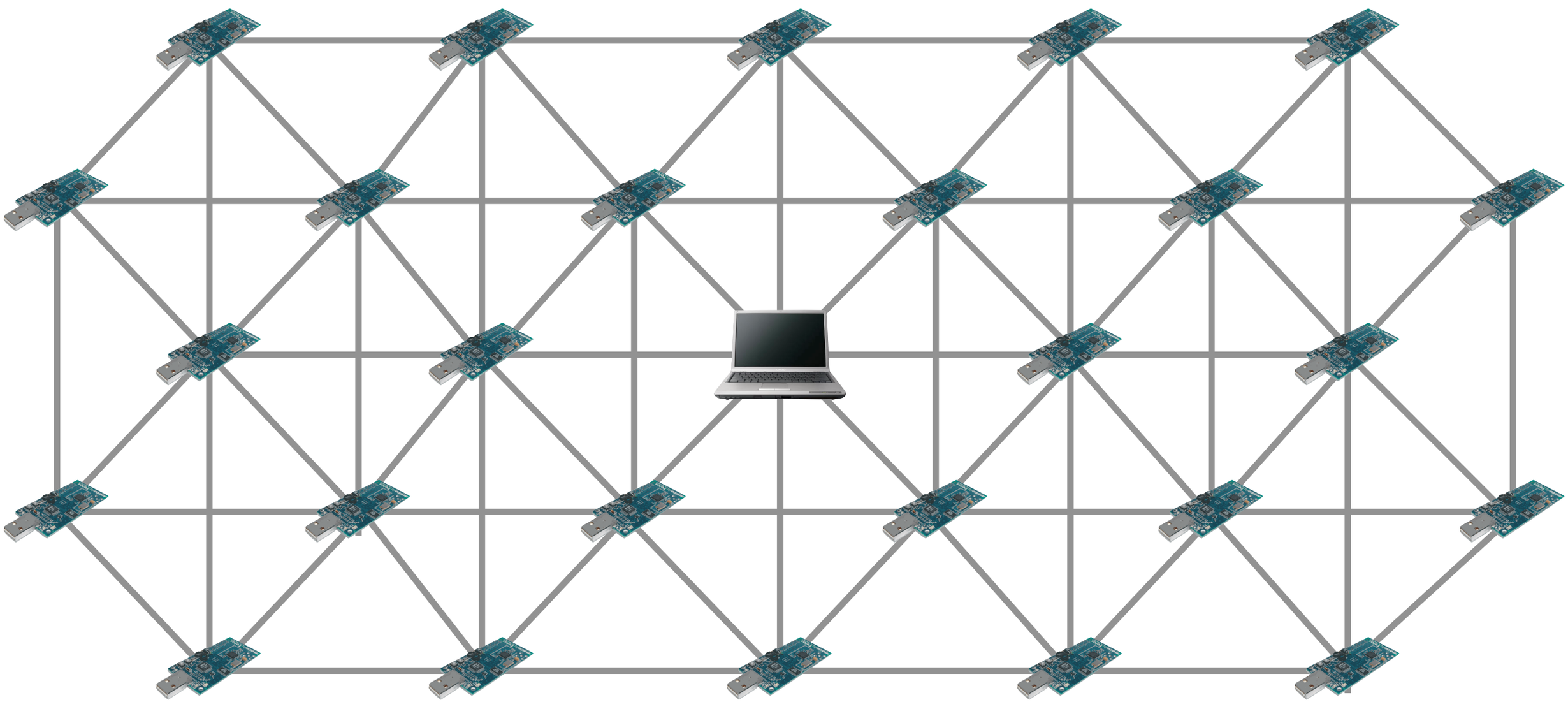
Topology and Logic Matter



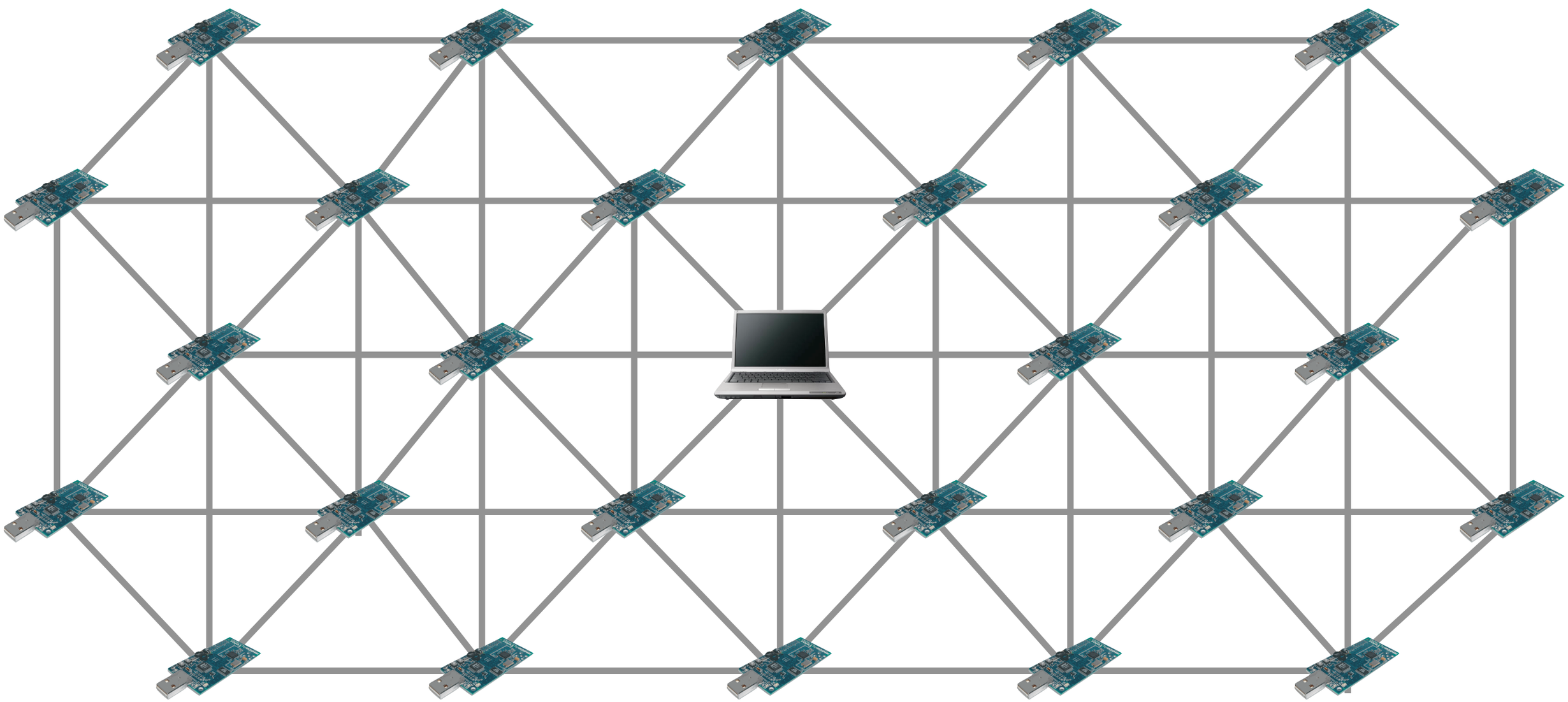
Topology and Logic Matter



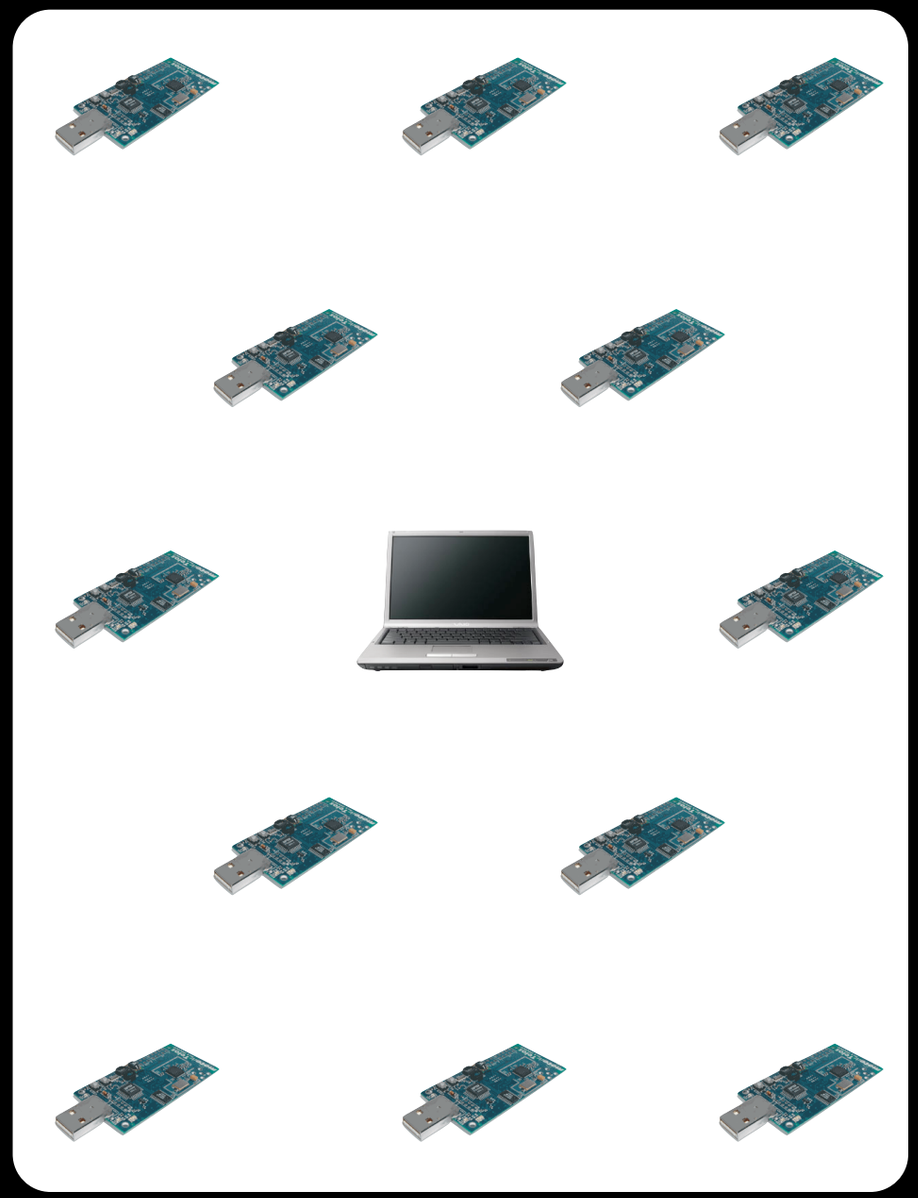
Topology and Logic Matter



Topology and Logic Matter



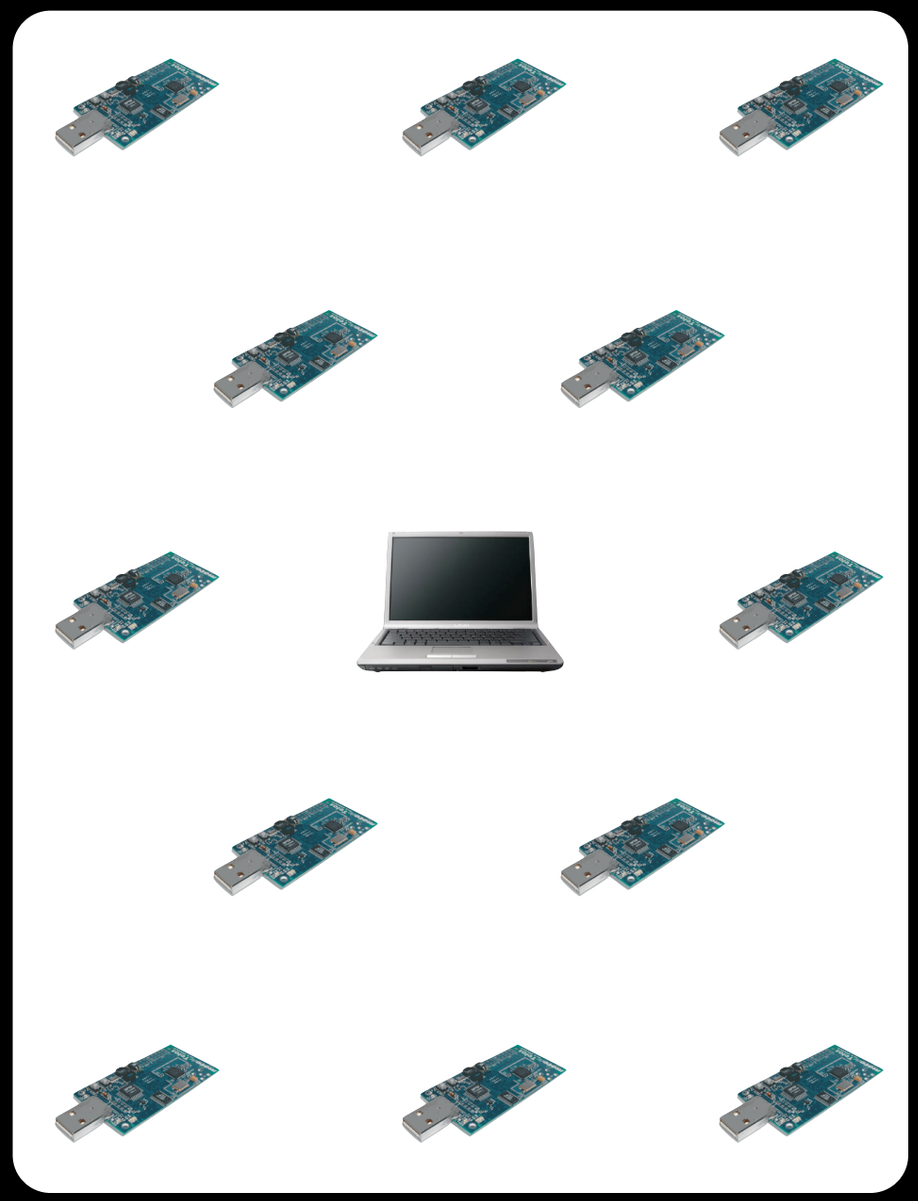
Solution



Solution

lightValues = lightSensor.sense()

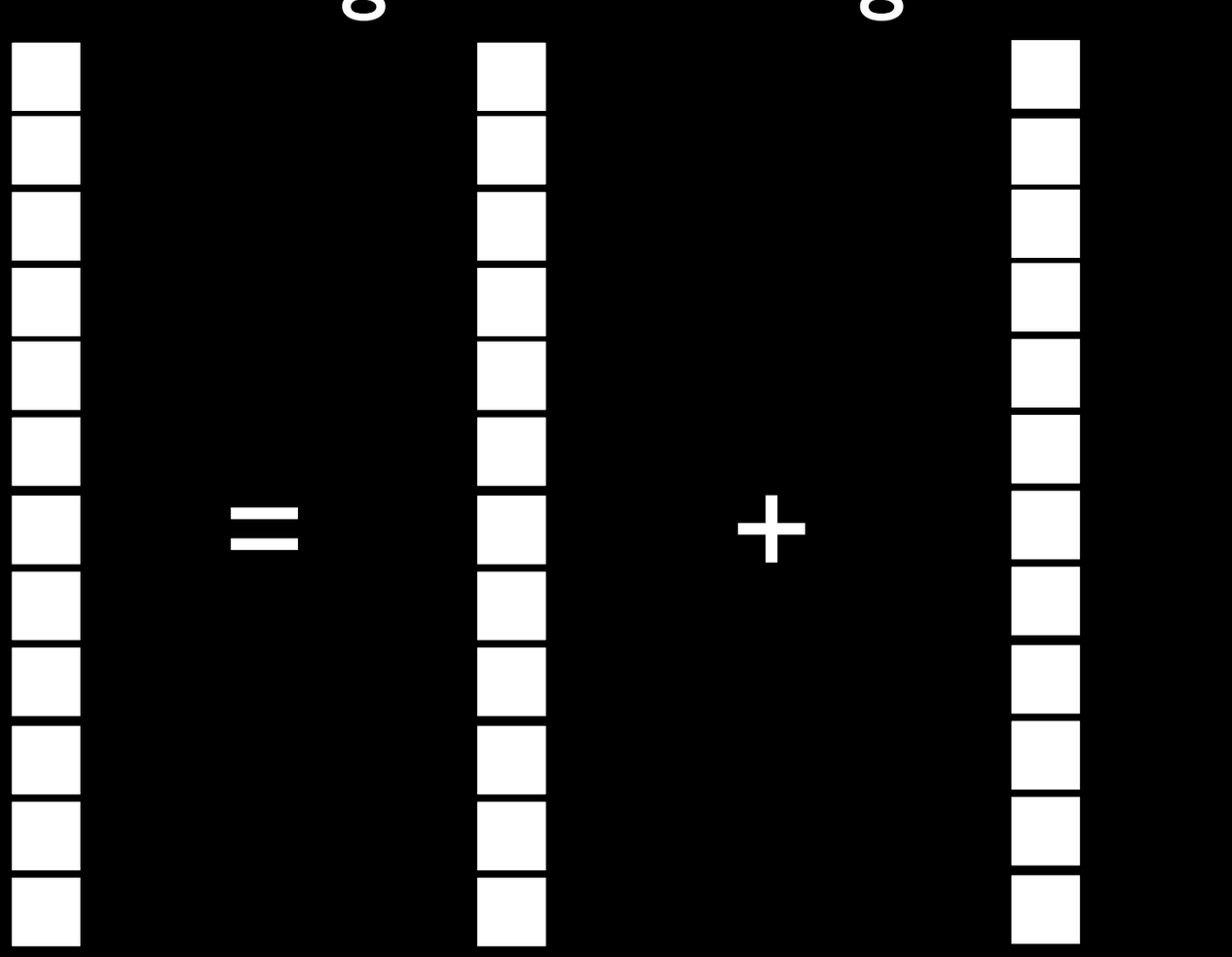
lightValues

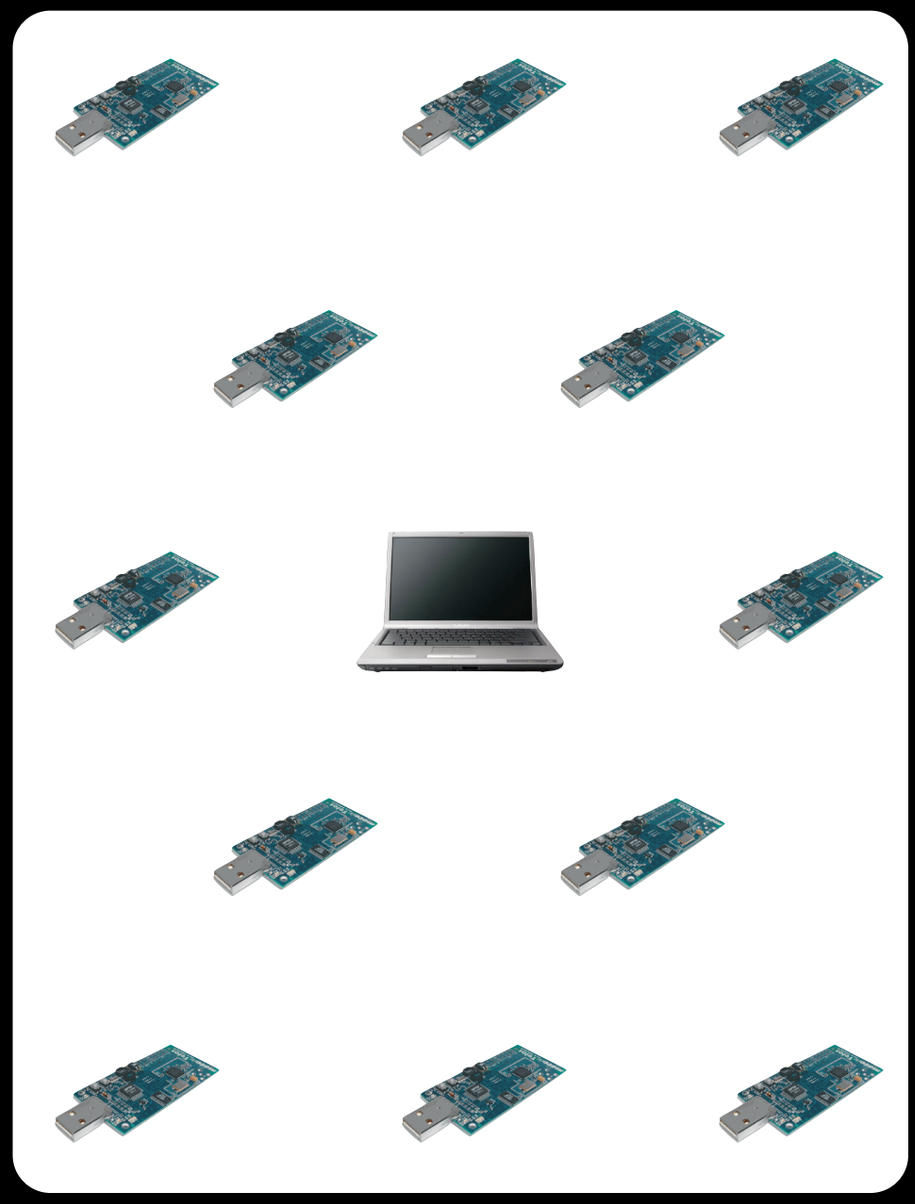


Solution

lightValues = lightSensor.sense()
lightOffsets = [1,35, ... 23]
X = lightOffsets + lightValues

X lightOffsets lightValues

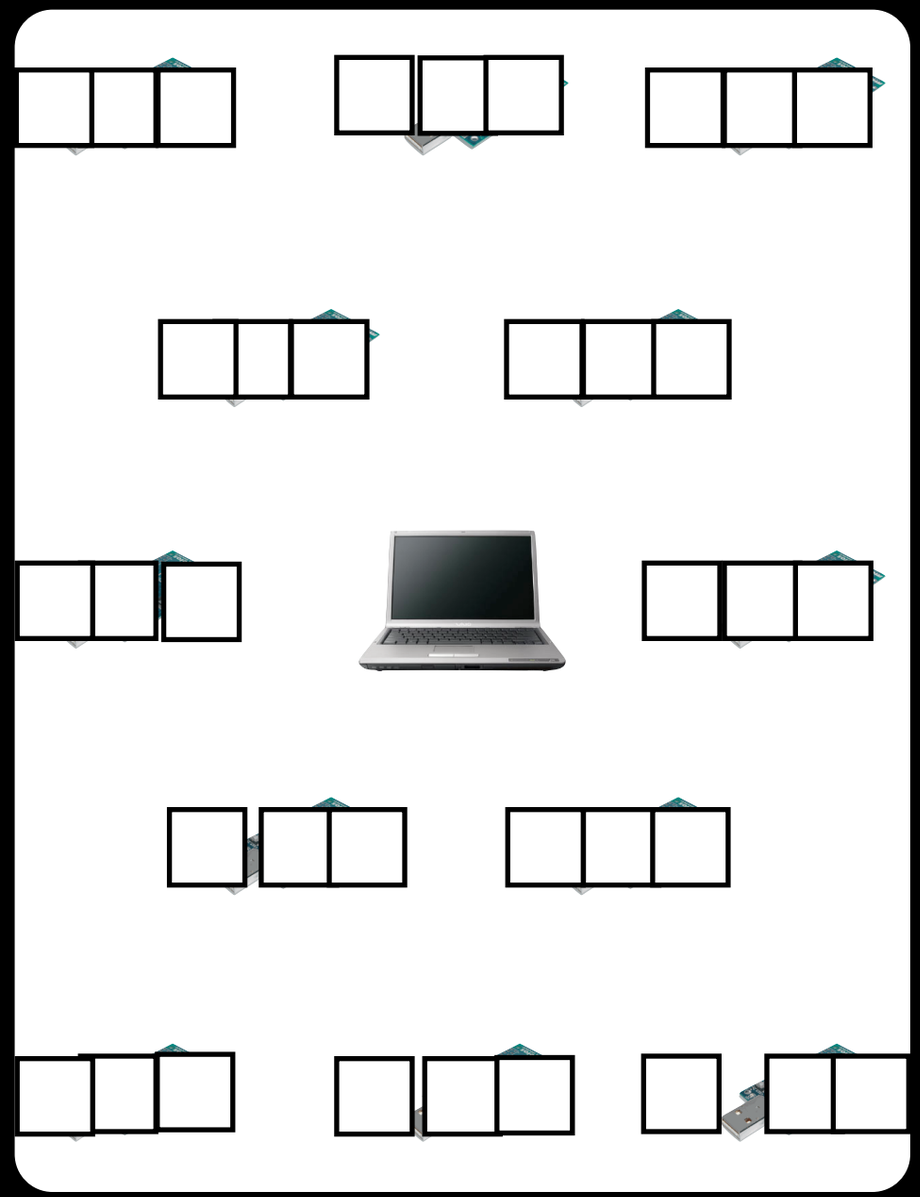




Solution

lightValues = lightSensor.sense()
lightOffsets = [1,35, ... 23]
X = lightOffsets + lightValues

X lightOffsets lightValues



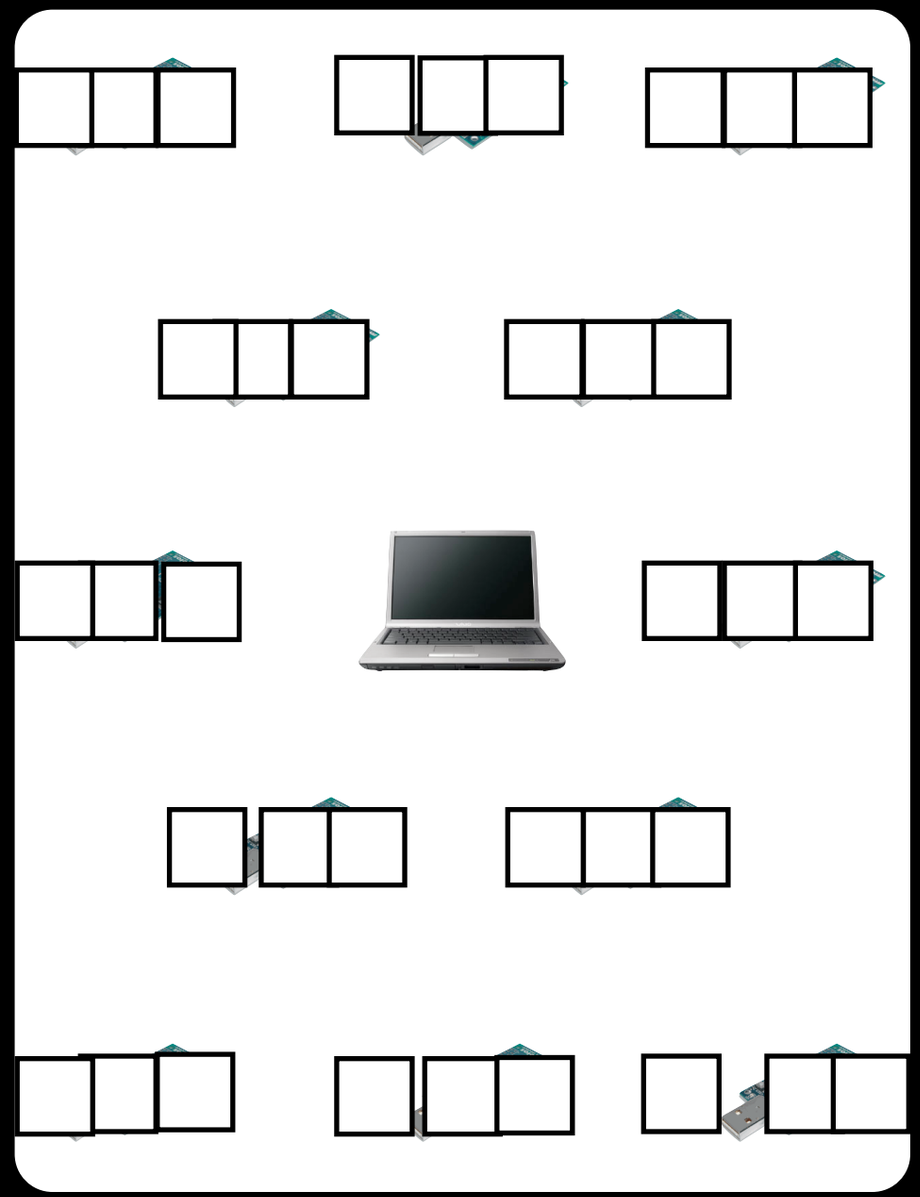
Solution

lightValues = lightSensor.sense()

lightOffsets = [1,35, ... 23]

X = lightOffsets + avg(lightValues)

X lightOffsets lightValues



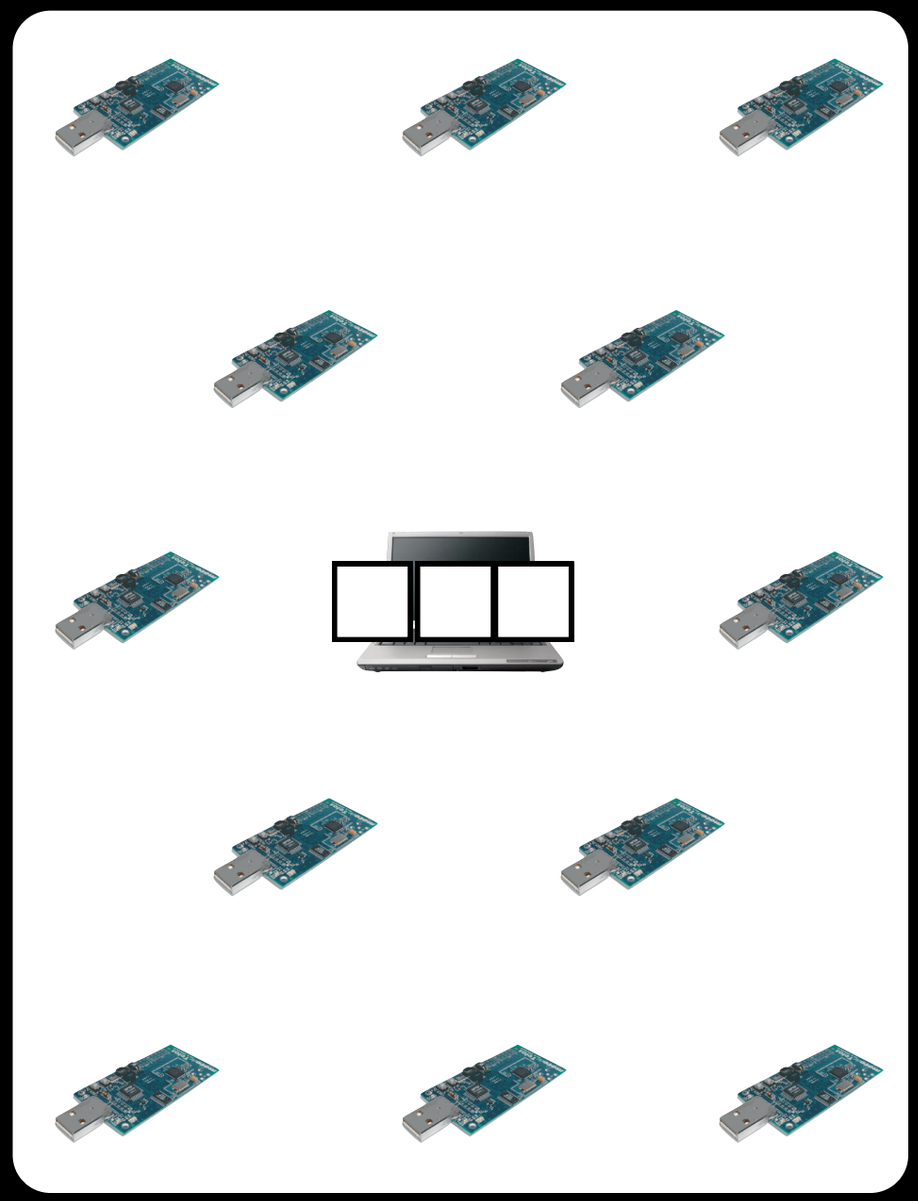
Solution

lightValues = lightSensor.sense()

lightOffsets = [1,35, ... 23]

X = lightOffsets + avg(lightValues)

X lightOffsets lightValues



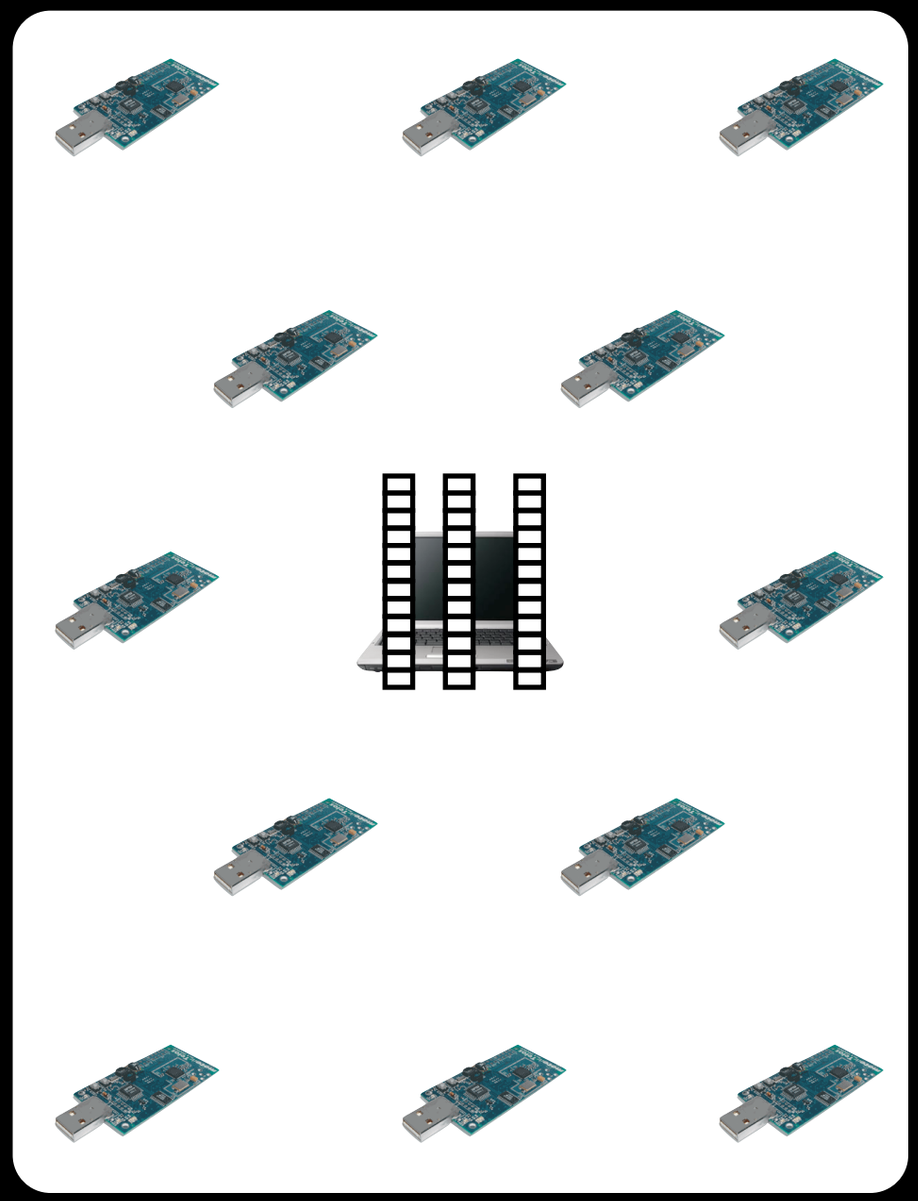
Solution

lightValues = lightSensor.sense()

lightOffsets = [1,35, ... 23]

X = lightOffsets + avg(lightValues)

X lightOffsets lightValues



Contributions

Contributions

- Easy to use programming abstraction for scientists and engineers

Contributions

- Easy to use programming abstraction for scientists and engineers
- Automatically choose best decomposition

Outline

- **Programming Abstraction**
- Compilation
- Evaluation
- Conclusion

Macrovector

35	2	3
2	3	4
18	4	5
94	5	6
10	6	7
61	7	8

A

Macrovector

35	2	3
2	3	4
18	4	5
94	5	6
10	6	7
61	7	8

A

+

35	8	4
61	2	3
10	6	2
94	1	7
2	9	3
18	9	10

B

Macrovector

35	10	7
2	12	7
18	13	15
94	6	13
10	12	9
61	9	11

=

35	2	3
2	3	4
18	4	5
94	5	6
10	6	7
61	7	8

+

35	8	4
2	9	3
18	9	10
94	1	7
10	6	2
61	2	3

C A B

Macrovector

35	16	12
2	6	12
18	24	10
94	5	42
10	54	21
61	63	80

C

=

35	2	3
2	3	4
18	4	5
94	5	6
10	6	7
61	7	8

A

.*

35	8	4
2	9	3
18	9	10
94	1	7
10	6	2
61	2	3

B

Macrovector

35	8	4
2	9	3
18	9	10
94	1	7
10	6	2
61	2	3

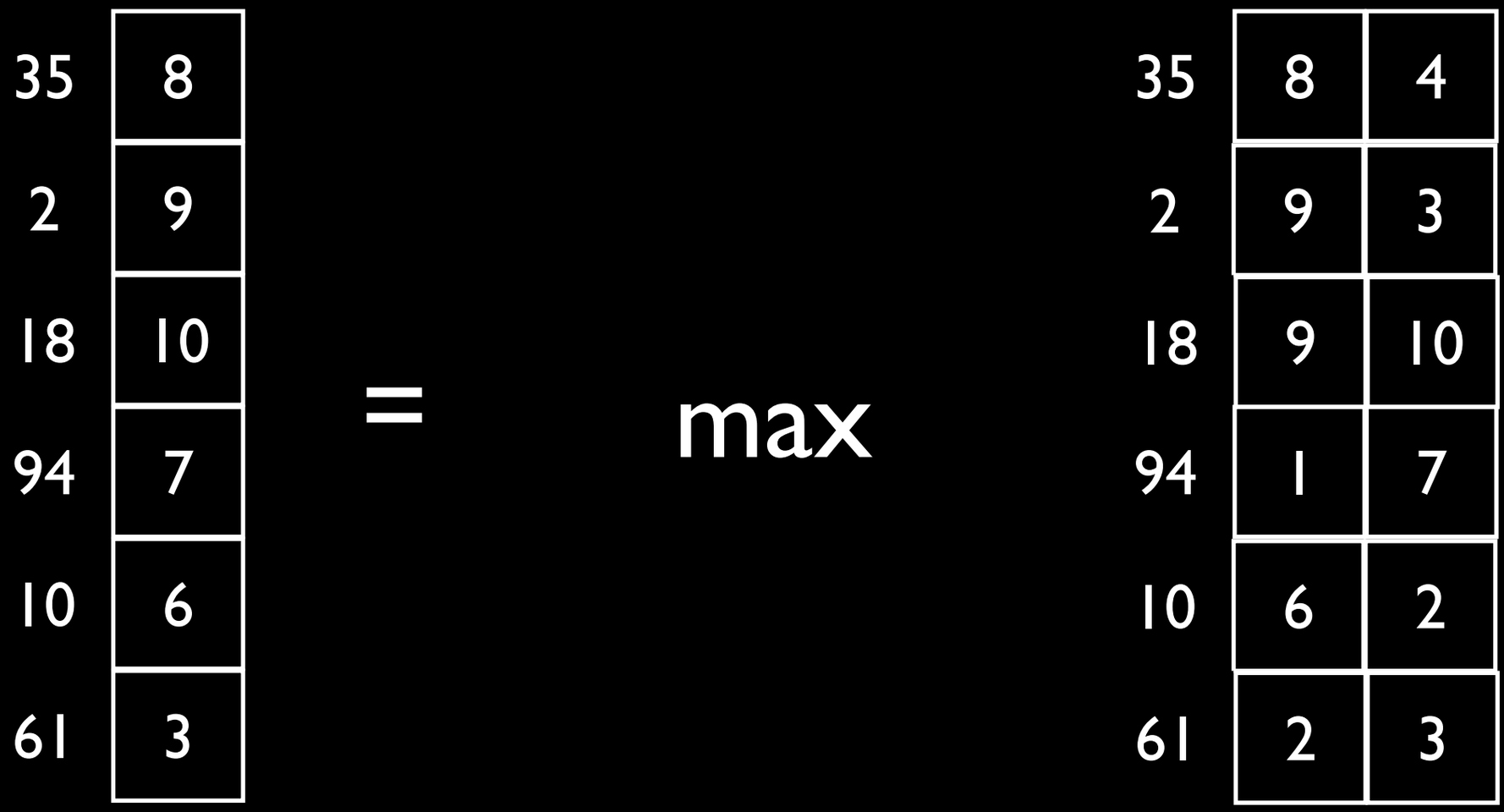
B

=

max

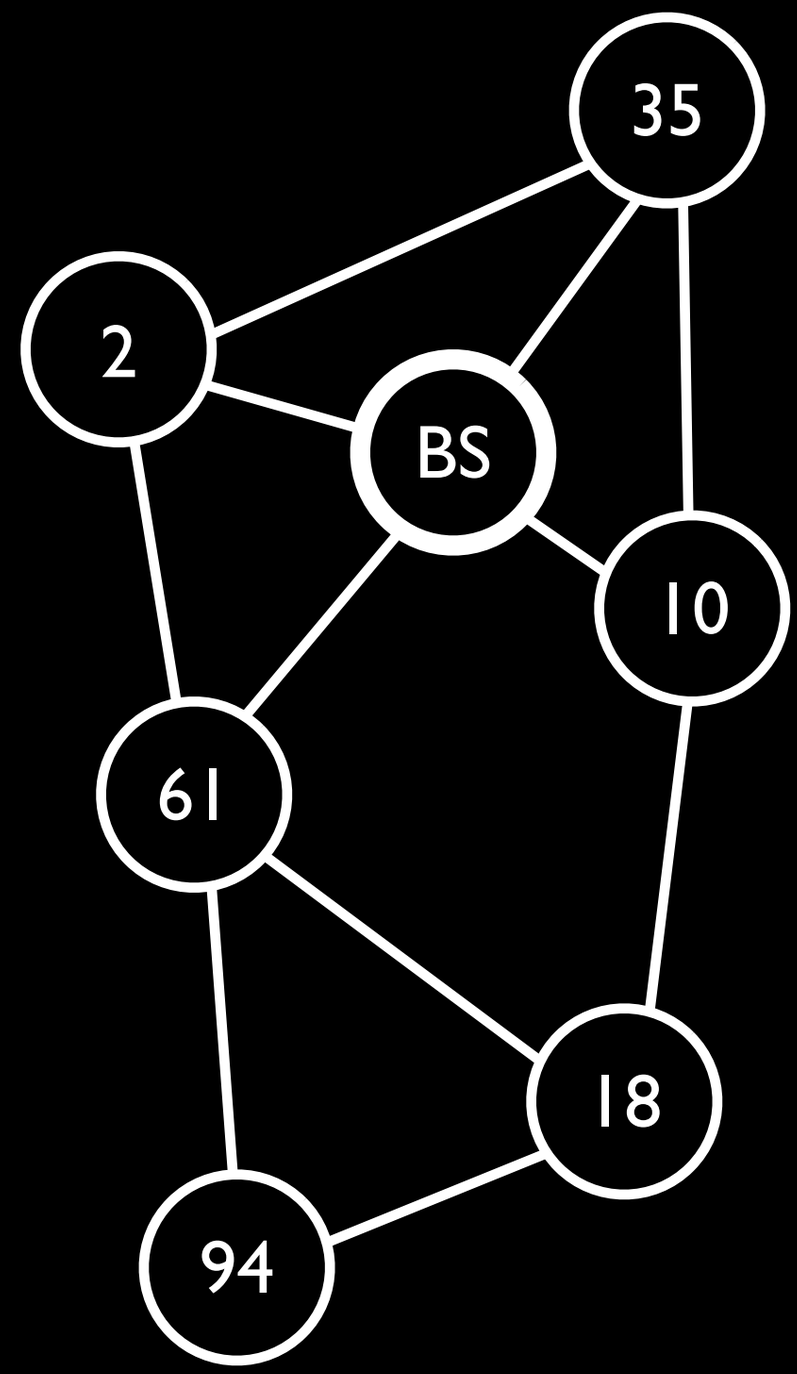
C

Macrovector

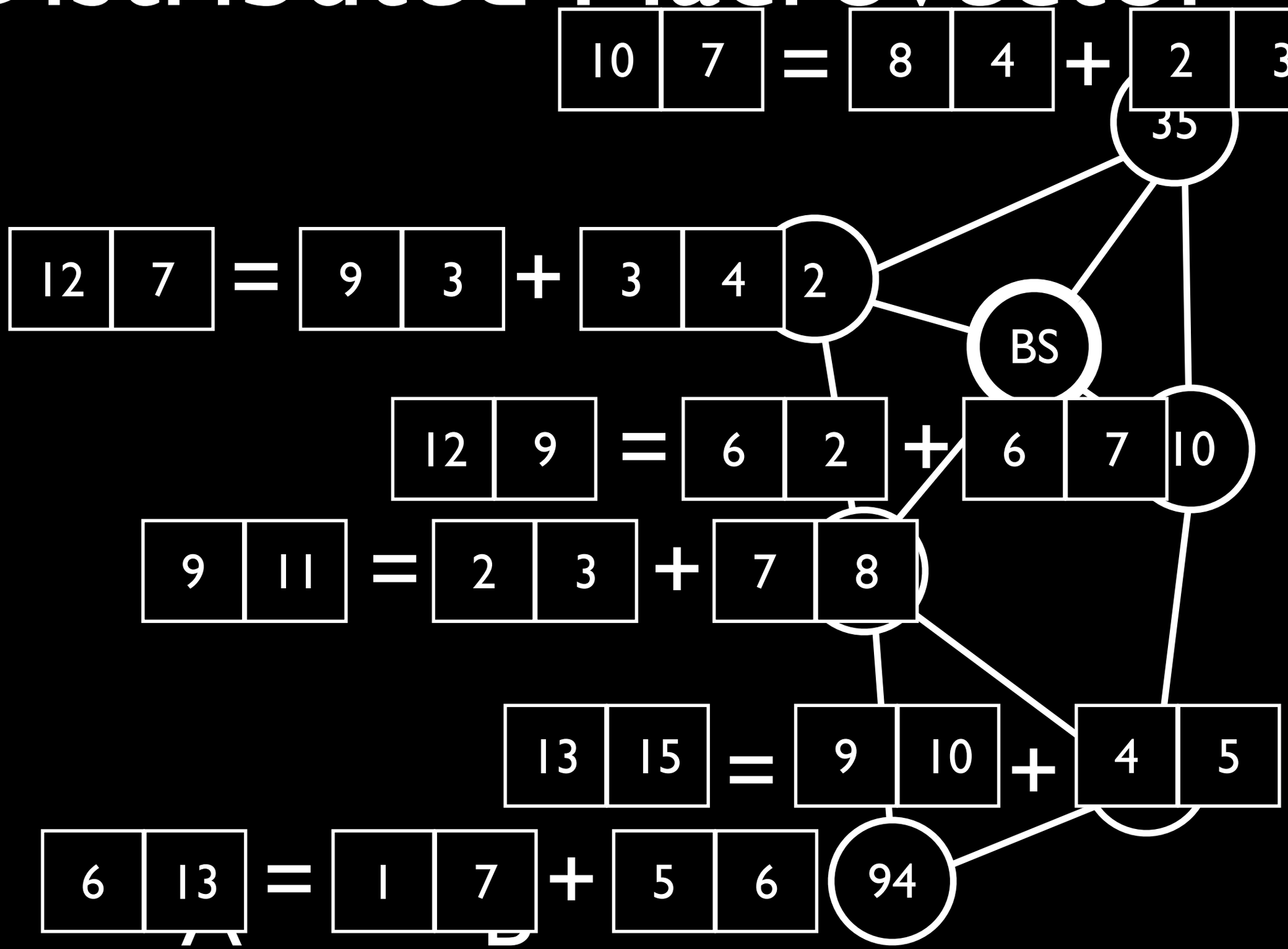


Distributed Macrovector

35	10	7		8	4		2	3
2	12	7		9	3		3	4
18	13	15	=	9	10	+	4	5
94	6	13		1	7		5	6
10	12	9		6	2		6	7
61	9	11		2	3		7	8
	C			A			B	



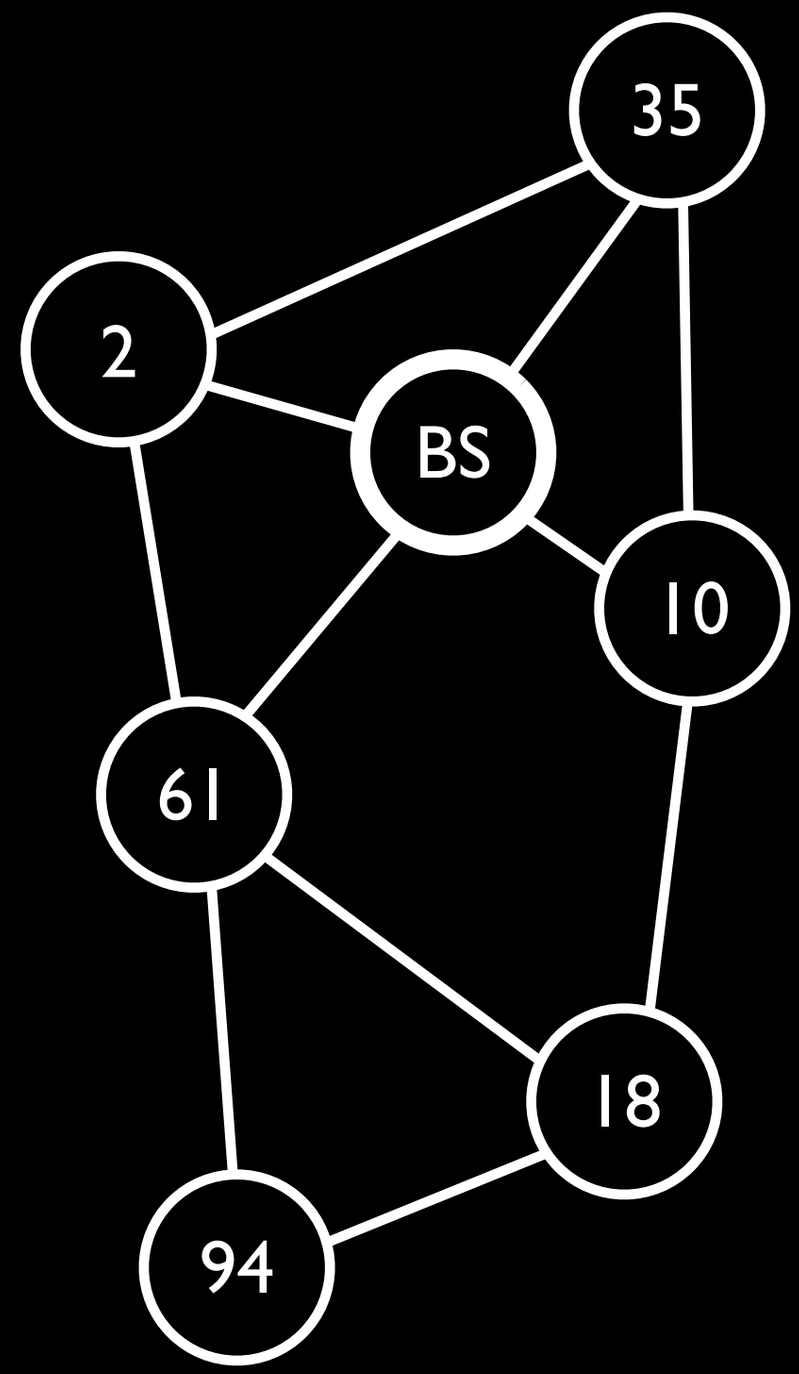
Distributed Macrovector



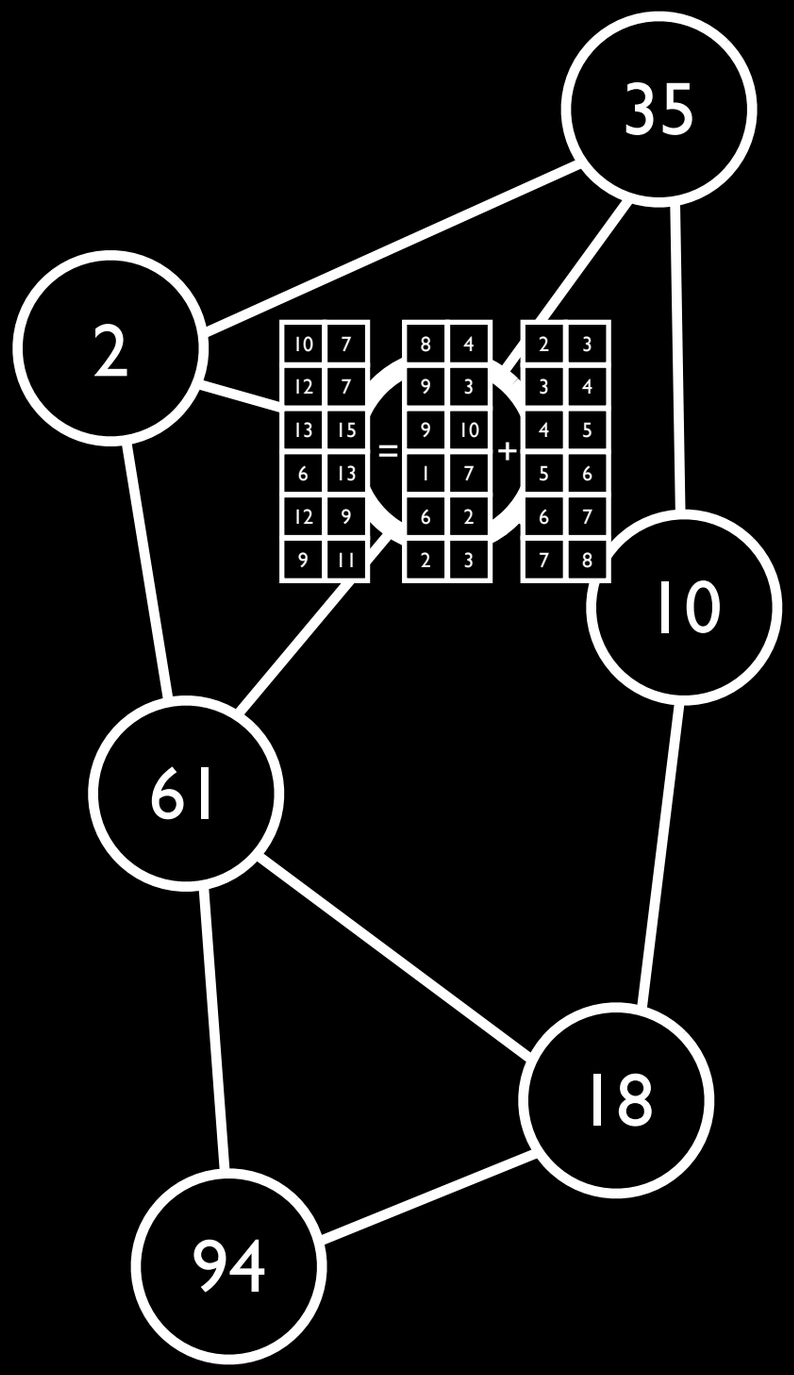
C

Centralized Macrovector

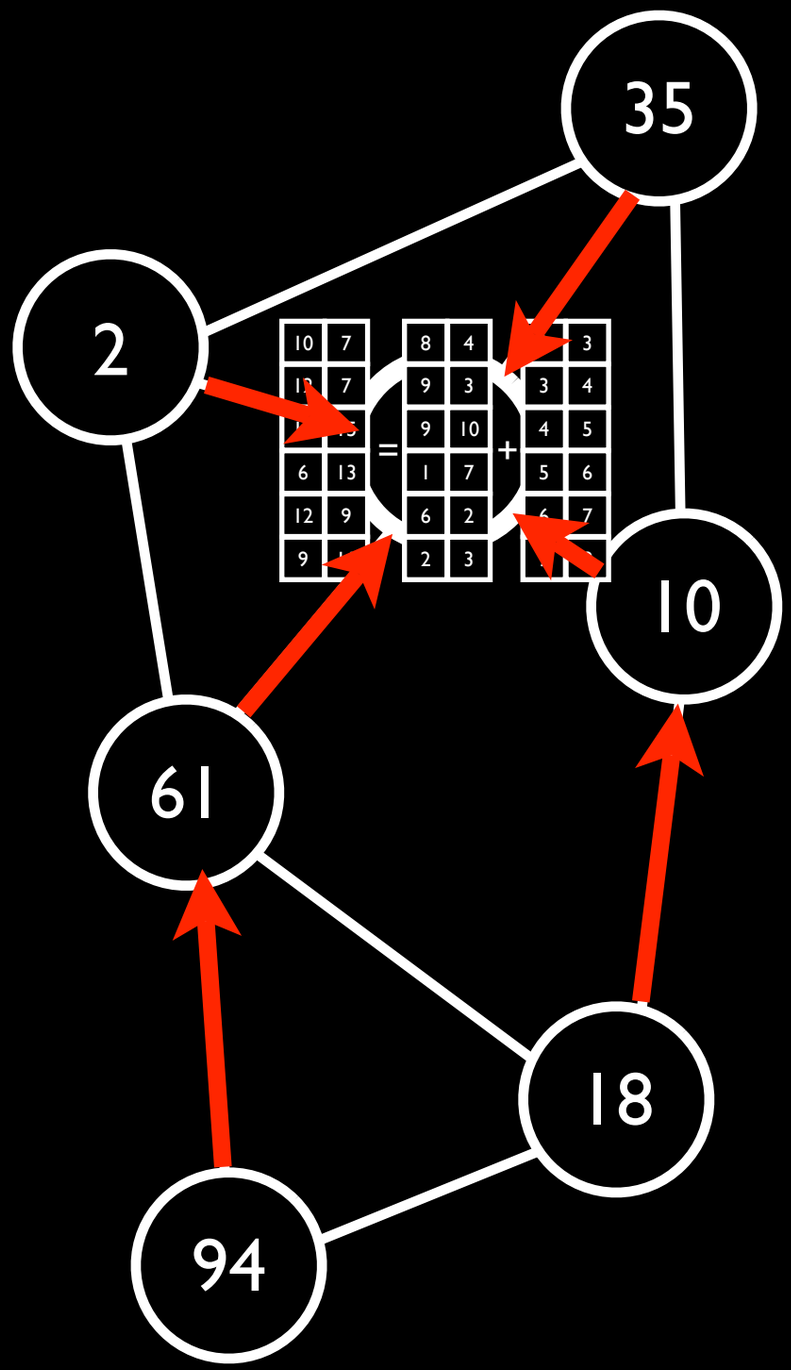
	C			A			B	
35	10	7	=	8	4	+	2	3
2	12	7		9	3		3	4
18	13	15		9	10		4	5
94	6	13		1	7		5	6
10	12	9		6	2		6	7
61	9	11		2	3		7	8



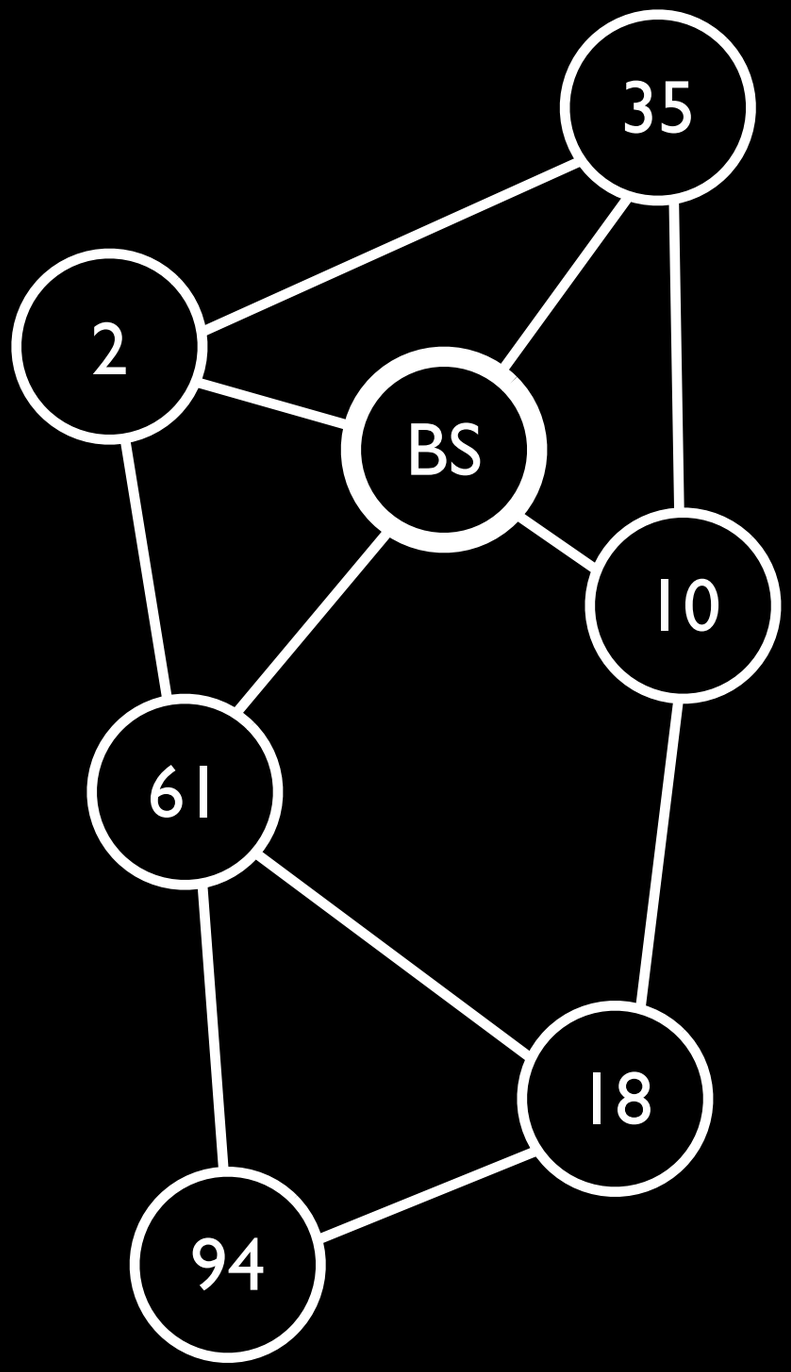
Centralized Macrovector

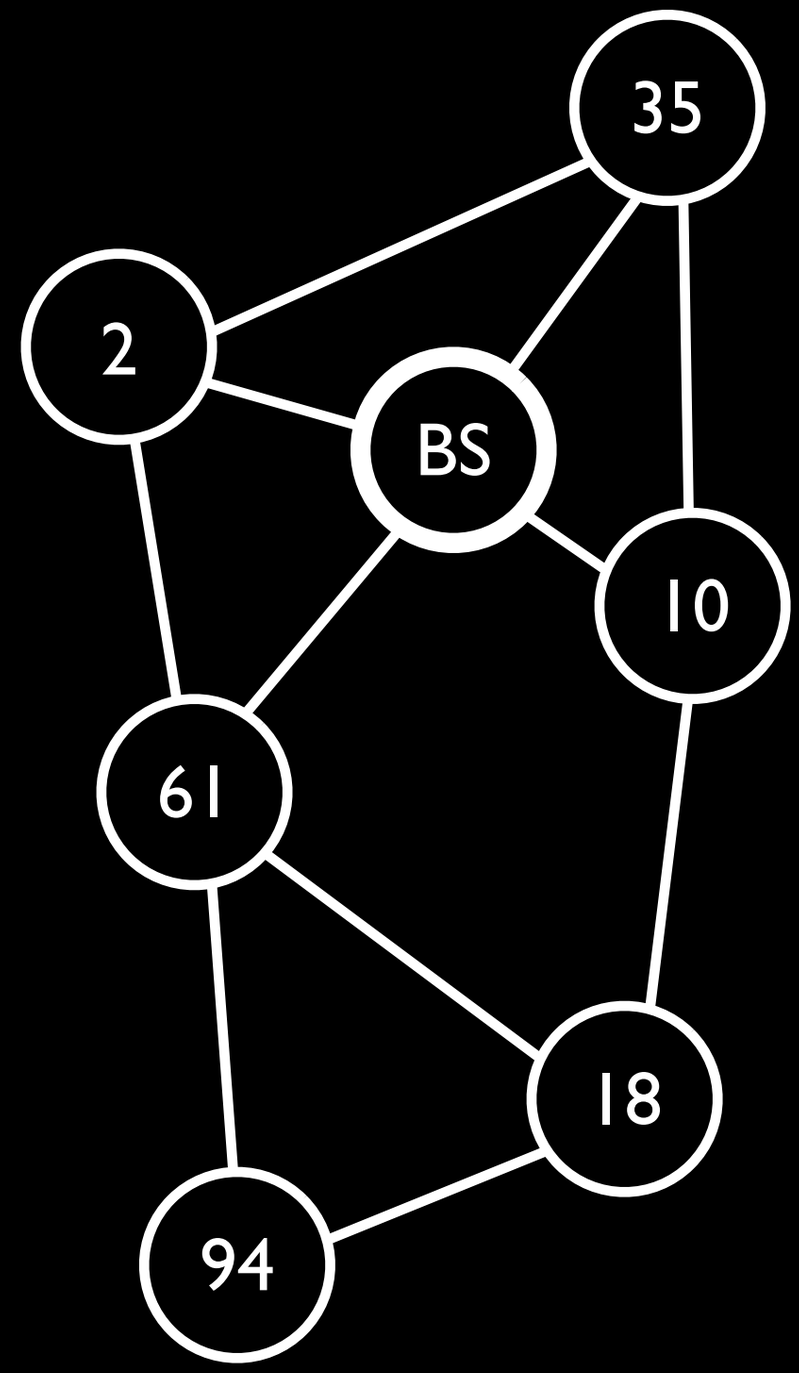


Centralized Macrovector

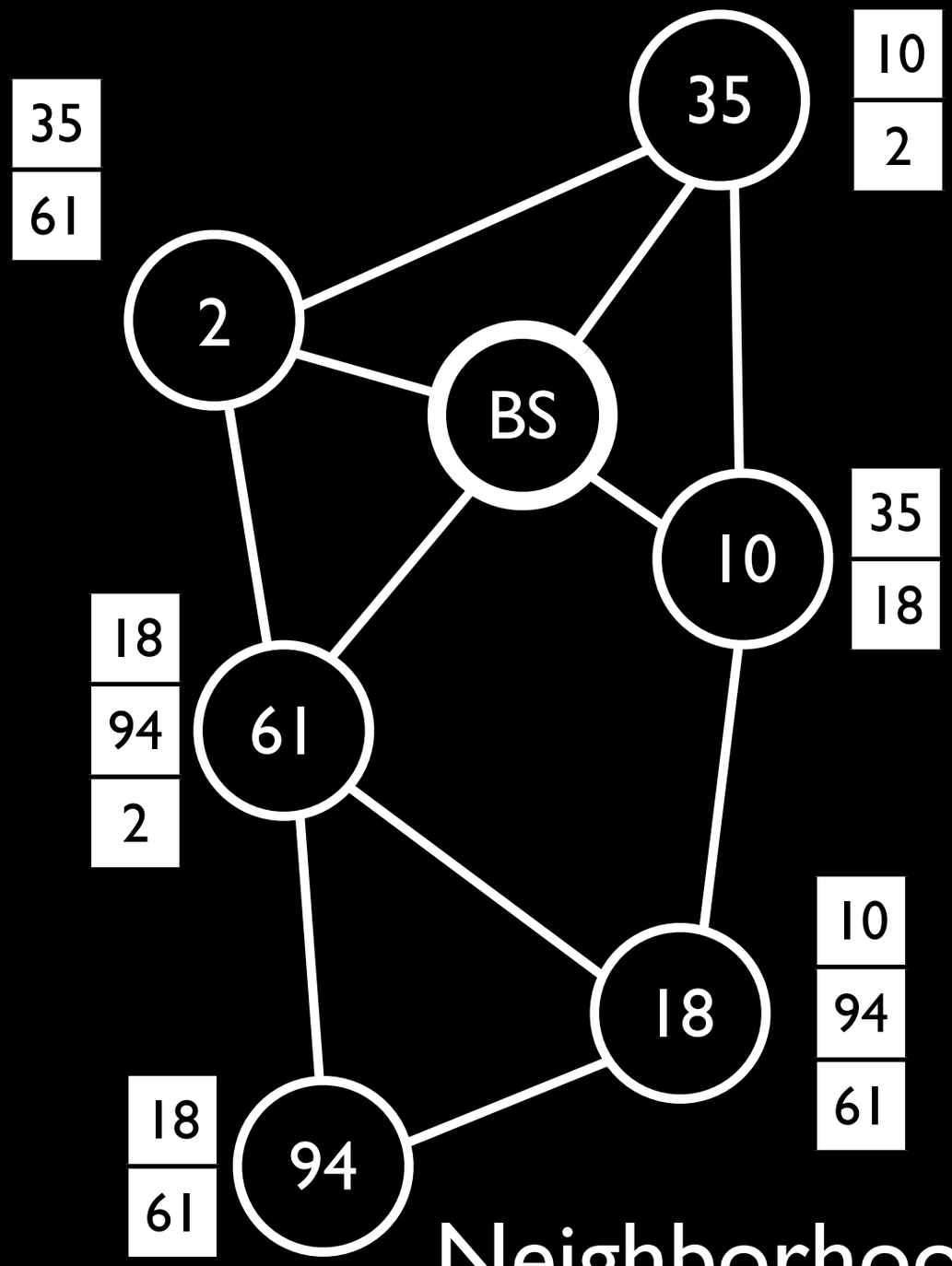


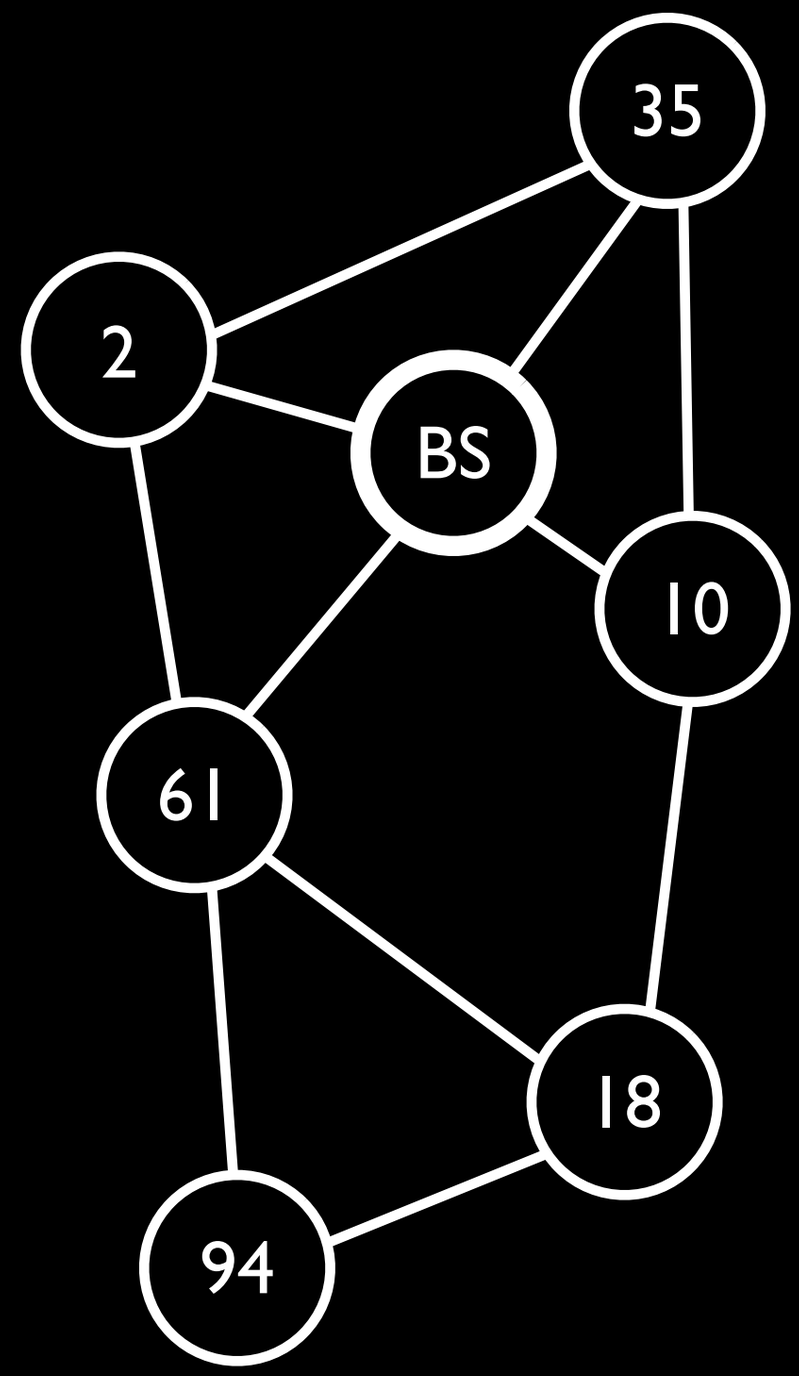
Other representations



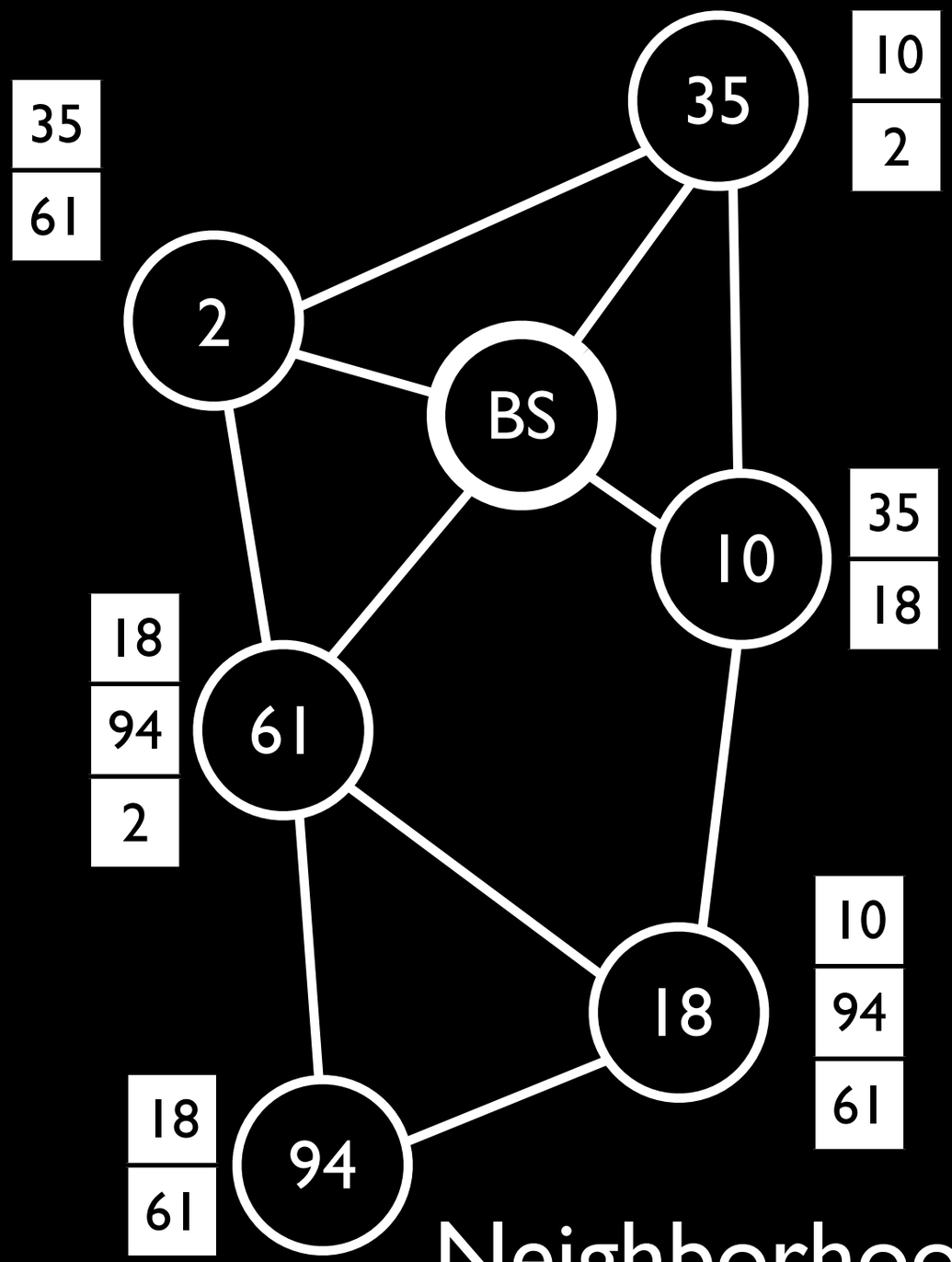


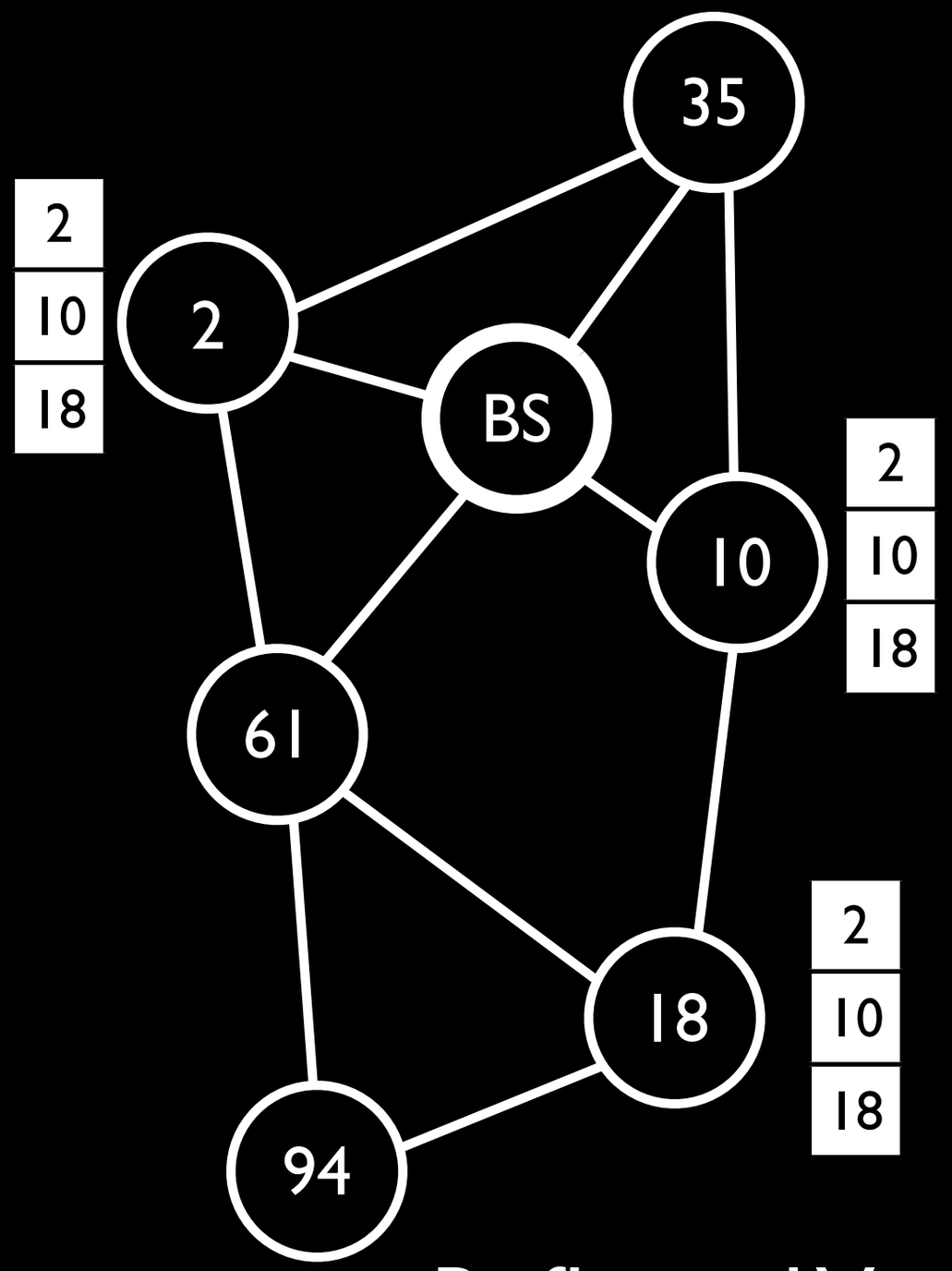
Other representations





Other representations





Outline

- Programming Abstraction
- **Compilation**
- Evaluation
- Conclusion

Sample Program

```
light = newMacrovector(motes)
offset = newMacrovector(motes)
X = newMacrovector(motes)

offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = light + offset;
end
```

Sample Program

```
distributed   light = newMacrovector(motes)
distributed   offset = newMacrovector(motes)
distributed   X = newMacrovector(motes)

offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = light + offset;
end
```

Sample Program

```
distributed    light = newMacrovector(motes)
centralized    offset = newMacrovector(motes)
distributed    X = newMacrovector(motes)

offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = light + offset;
end
```

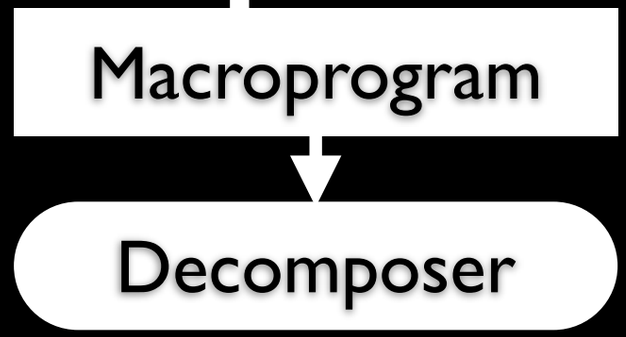
Sample Program

```
centralized    light = newMacrovector(motes)
centralized    offset = newMacrovector(motes)
centralized    X = newMacrovector(motes)

offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = light + offset;
end
```

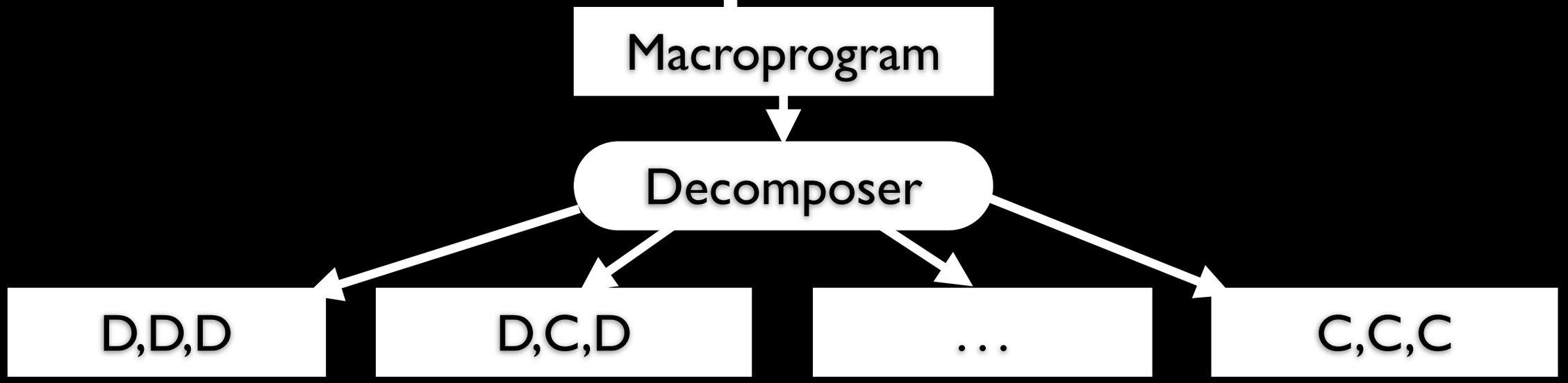
Compilation

Decomposer

Compilation



Compilation



Function Resolution

distributed light = newMacrovector(motes)
distributed offset = newMacrovector(motes)
centralized X = newMacrovector(motes)

```
offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = light + offset;
end
```

Function Library

plusddc.m

plusccc.m

plusddd.m

Function Resolution

distributed light = newMacrovector(motes)
distributed offset = newMacrovector(motes)
centralized X = newMacrovector(motes)

```
offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = light + offset;
end
```

Function Library

plusddc.m

plusccc.m

plusddd.m

Addition operation
input: distributed, distributed
output: centralized

Function Resolution

distributed light = newMacrovector(motes)
distributed offset = newMacrovector(motes)
centralized X = newMacrovector(motes)

```
offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = plusddc(light,offset);
end
```

Function Library

plusddc.m

plusccc.m

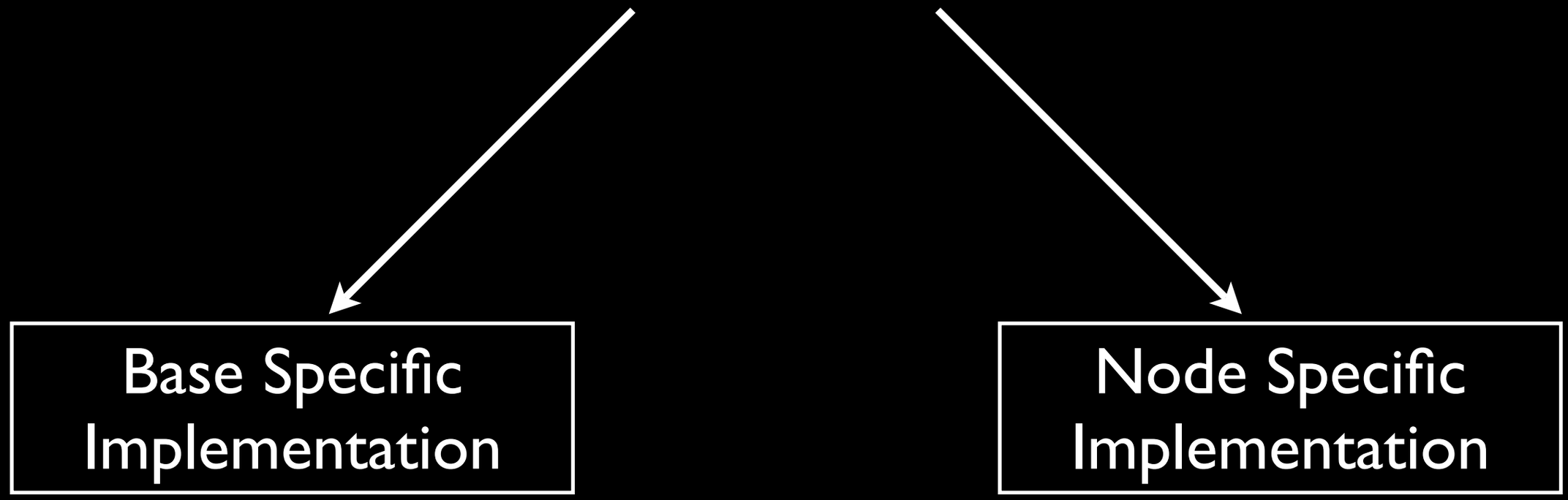
plusddd.m

Sample Program

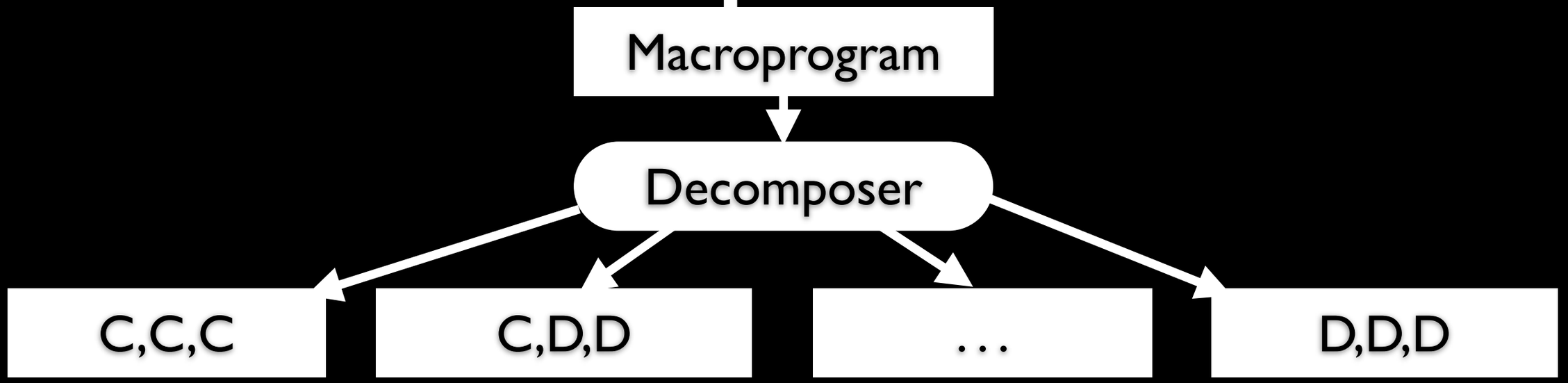
```
light = newMacrovector(motes)
offset = newMacrovector(motes)
X = newMacrovector(motes)

offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = plusddc(light,offset);
end
```

Sample Program

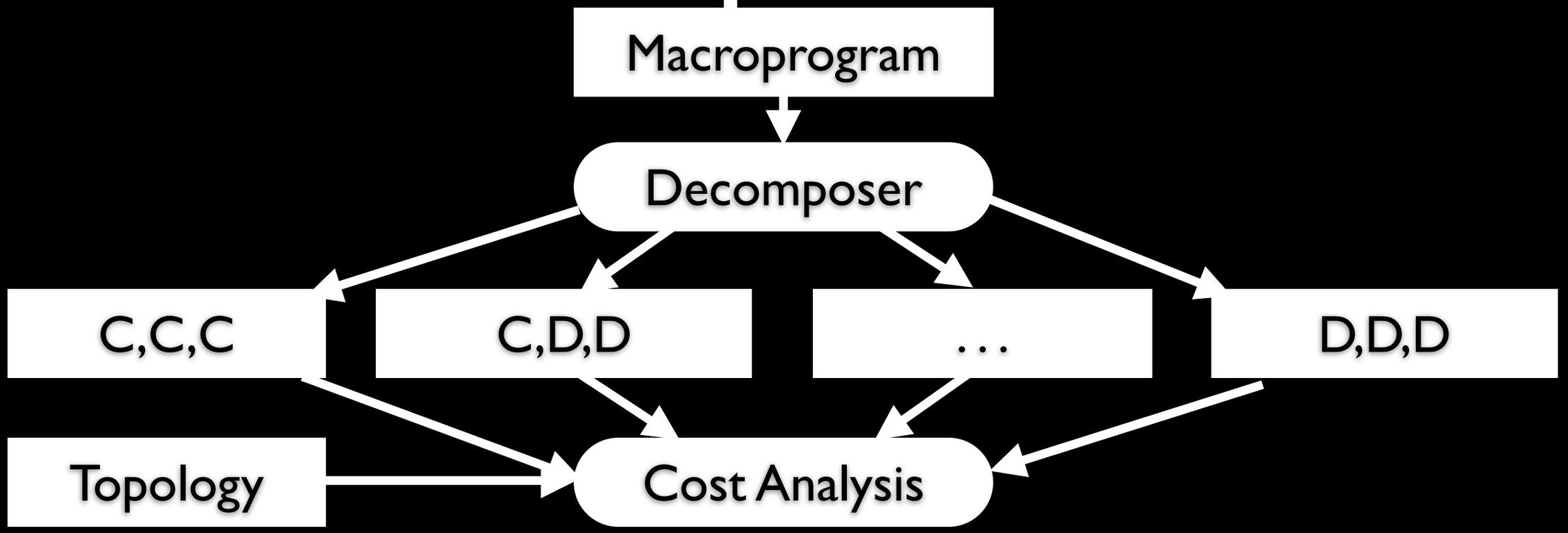
```
light = newMacrovector(motes)
offset = newMacrovector(motes)
X = newMacrovector(motes)

offset = [1,35,...,25];

every(uint16(10000))
  light = lightSensors.sense();
   X = plusddc(light,offset);
end
```



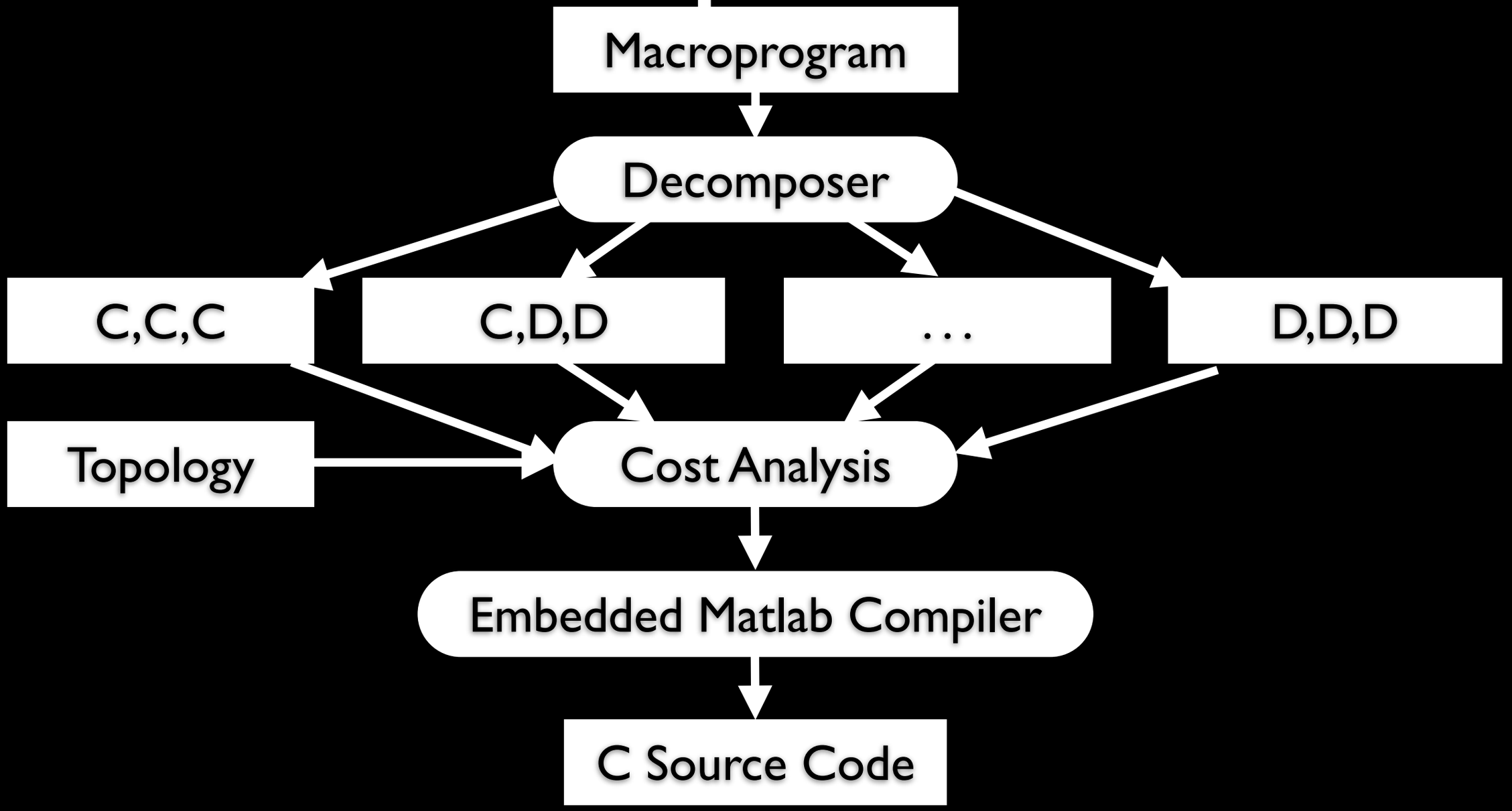
Compilation



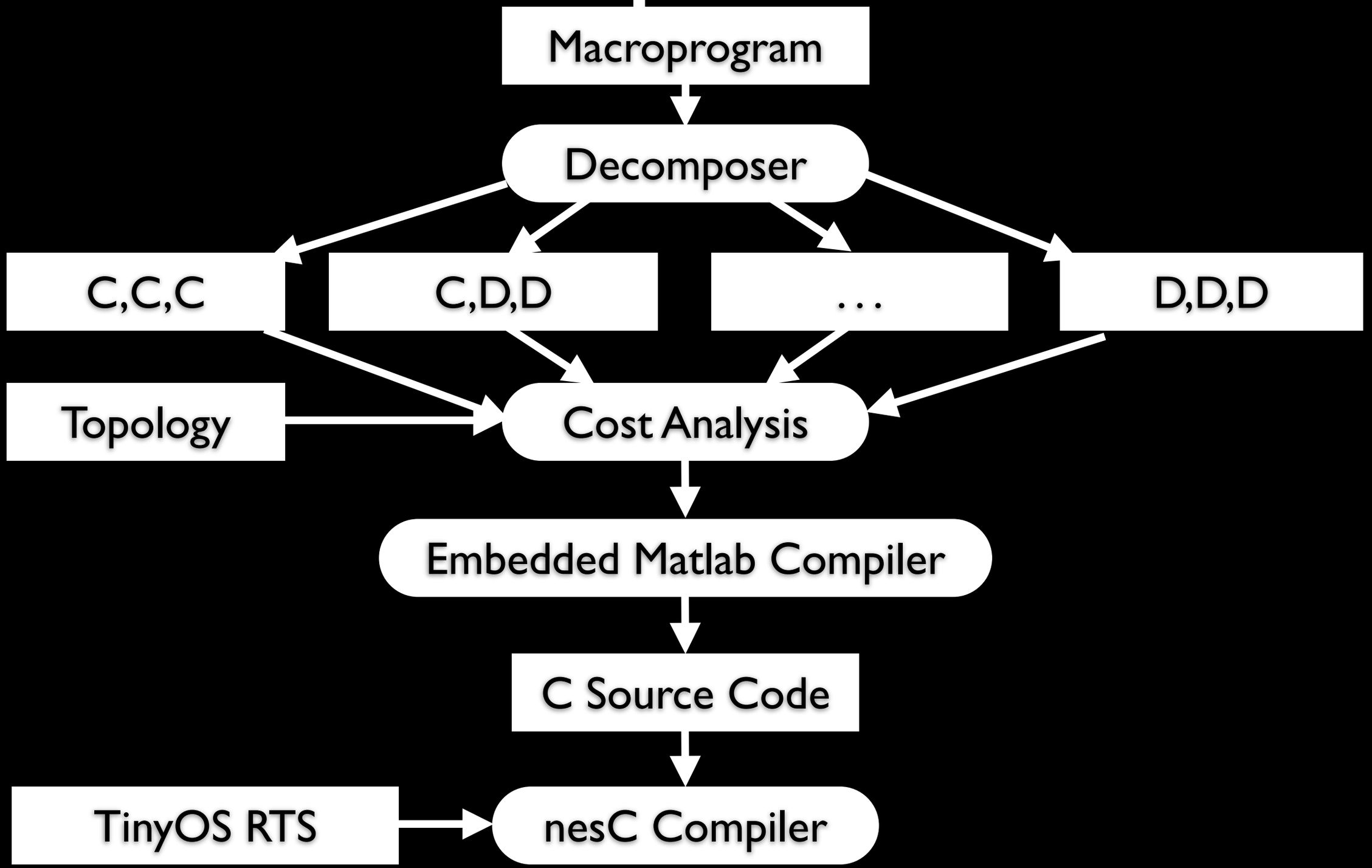
Compilation



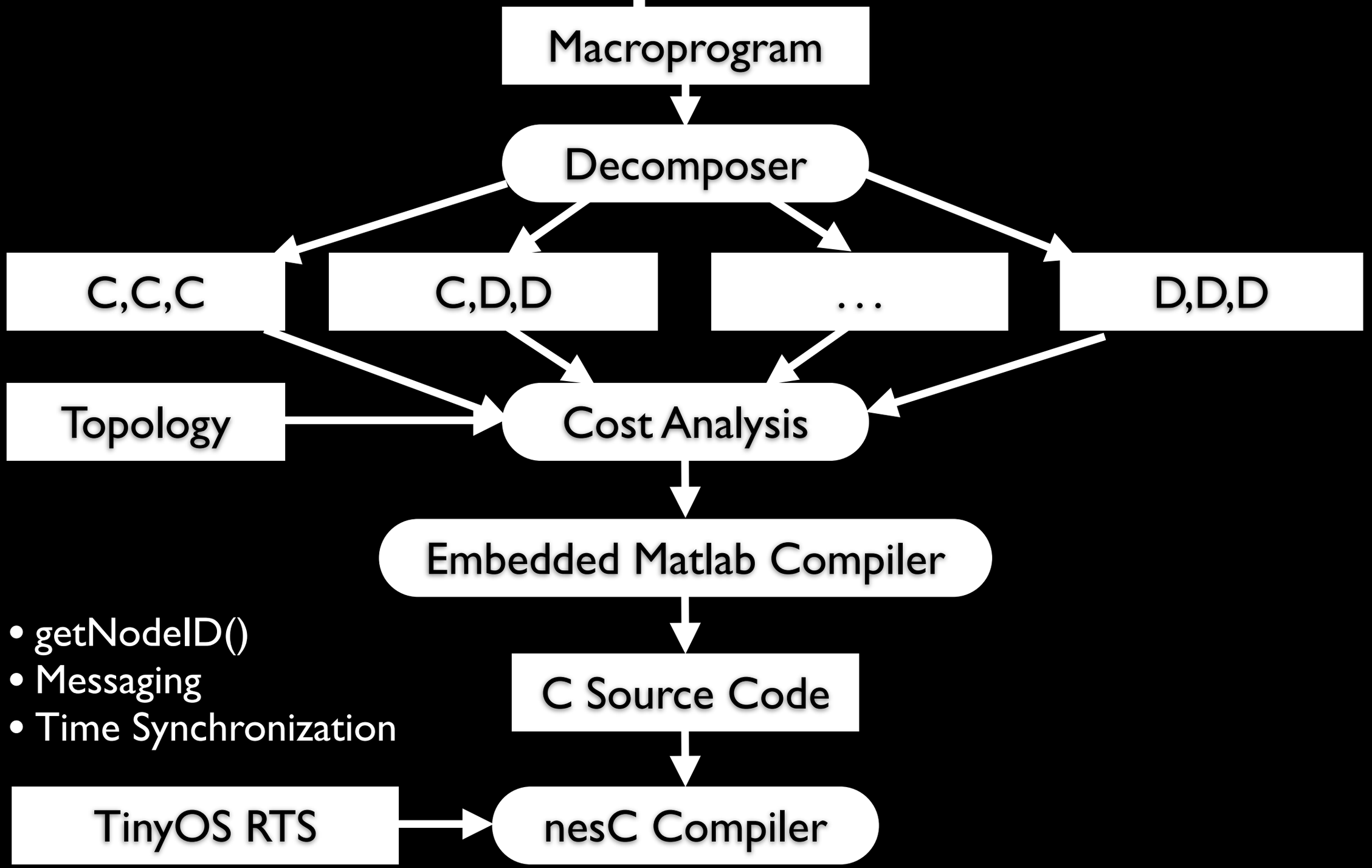
Compilation



Compilation

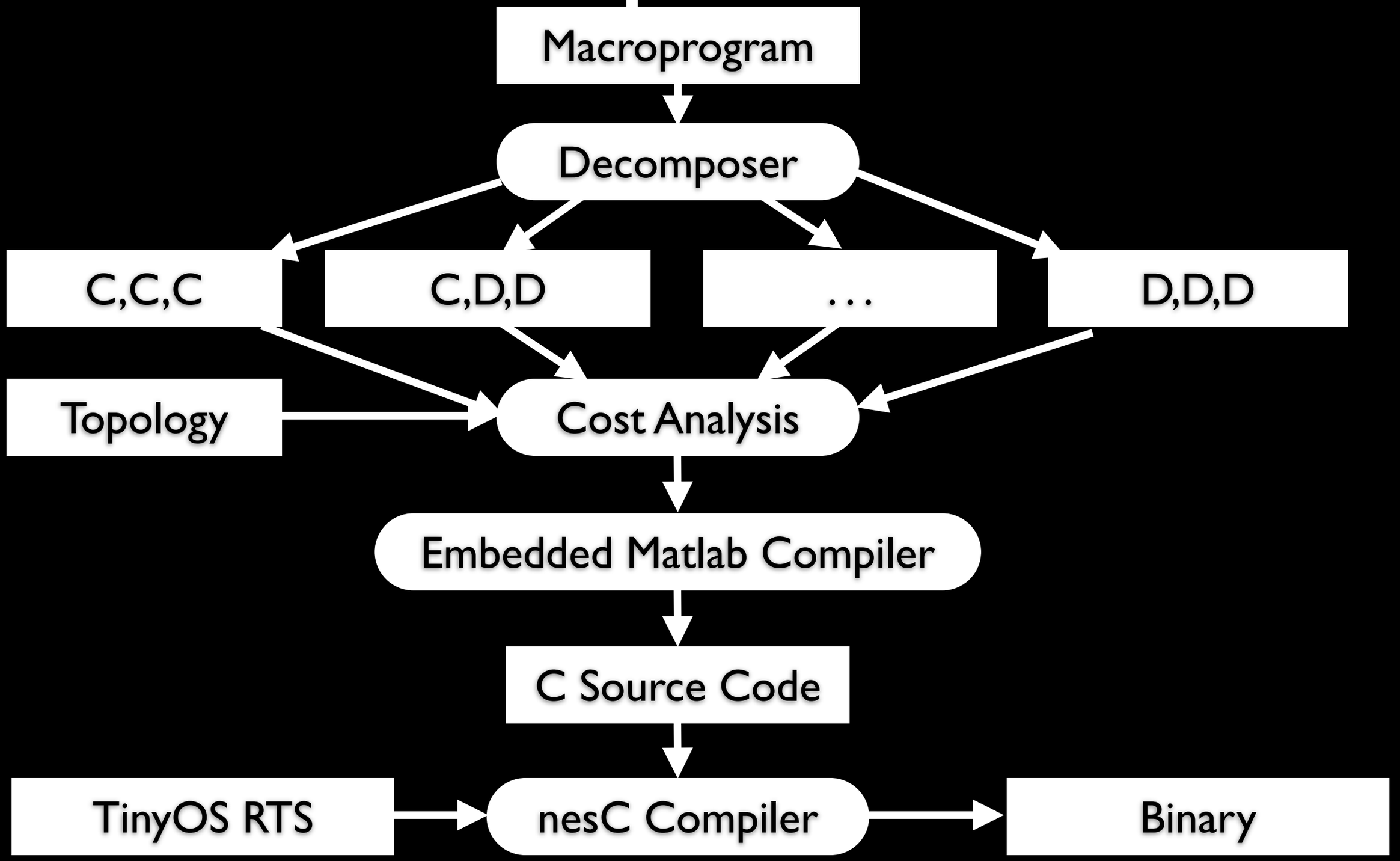


Compilation



- getNodeID()
- Messaging
- Time Synchronization

Compilation



Outline

- Programming Abstraction
- Compilation
- **Evaluation**
- Conclusion

Lines of Code

	nesC/TinyOS
Data Collection (Surge)	400
Tracking (PEG)	780

Lines of Code

	nesC/TinyOS	MacroLab
Data Collection (Surge)	400	7
Tracking (PEG)	780	19

Platform	ROM Size	RAM Size
TelosB	49,152	10,240
MICAz	131,072	4,096

Platform	ROM Size	RAM Size
TelosB	49,152	10,240
MICAz	131,072	4,096

Application	Program Size	Heap Size
SurgeTelos	24,790	911
PEG	61,440	3,072

Platform	ROM Size	RAM Size
TelosB	49,152	10,240
MICAz	131,072	4,096

Application	Program Size	Heap Size
SurgeTelos	24,790	911
PEG	61,440	3,072
MacroLab_Surge	**19,374**	**669**
MacroLab_PEG	**18,536**	**770**

Platform	ROM Size	RAM Size
TelosB	49,152	10,240
MICAz	131,072	4,096

Application	Program Size	Heap Size
SurgeTelos	24,790	911
PEG	61,440	3,072
MacroLab_Surge	**19,374**	**669**
MacroLab_PEG	**18,536**	**770**
Blink	2,472	38
CountRadio	11,266	351
Oscilloscope	9,034	335
OscilloscopeRF	14,536	449
SenseToRfm	14,248	403
TOSBase	10,328	1,827

Platform	ROM Size	RAM Size
TelosB	49,152	10,240
MICAz	131,072	4,096

Application	Program Size	Heap Size
SurgeTelos	24,790	911
PEG	61,440	3,072
MacroLab_Surge	**19,374**	**669**
MacroLab_PEG	**18,536**	**770**

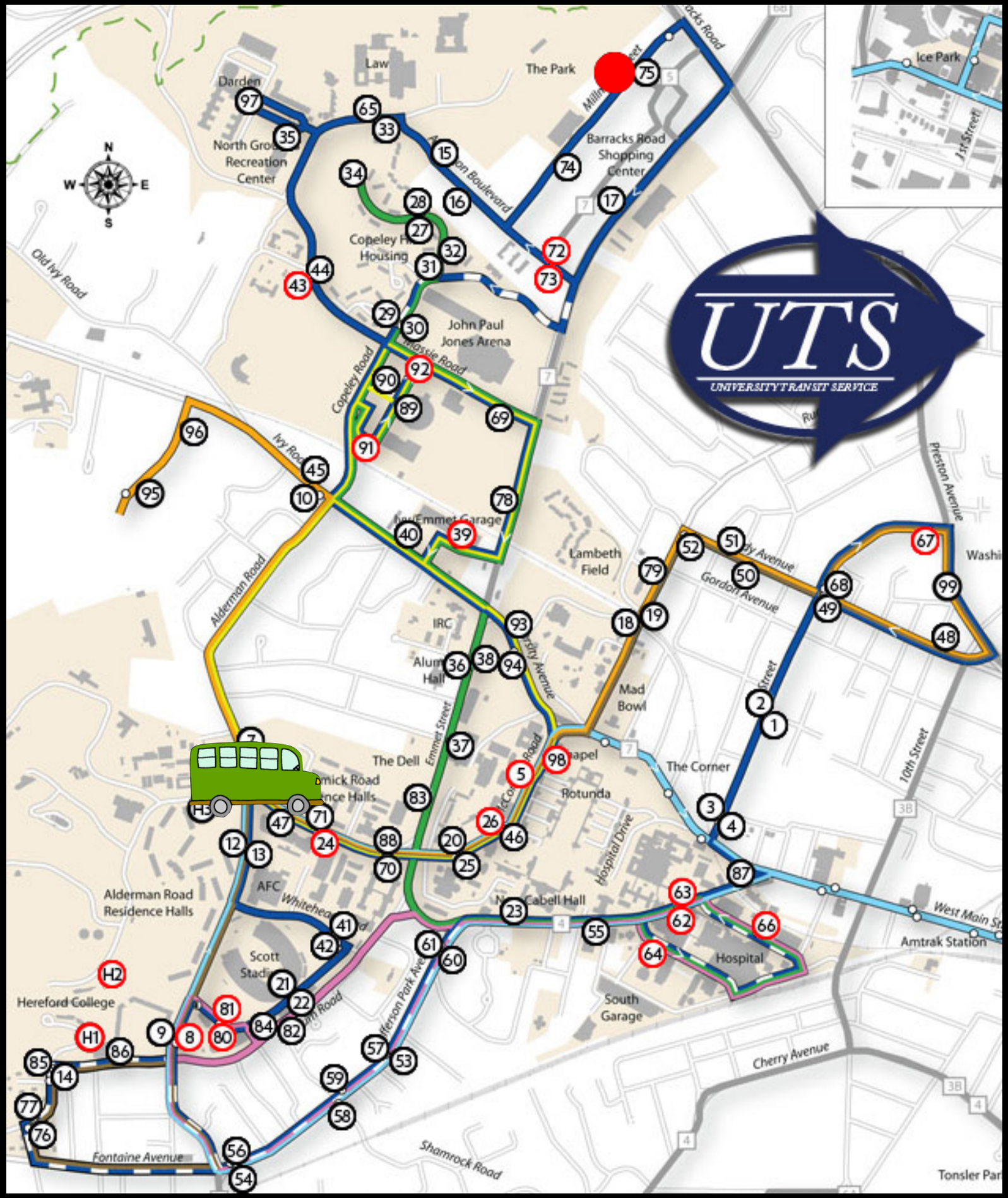
MacroLab_Surge	**1,822**	**191**
MacroLab_PEG	**1,702**	**90**

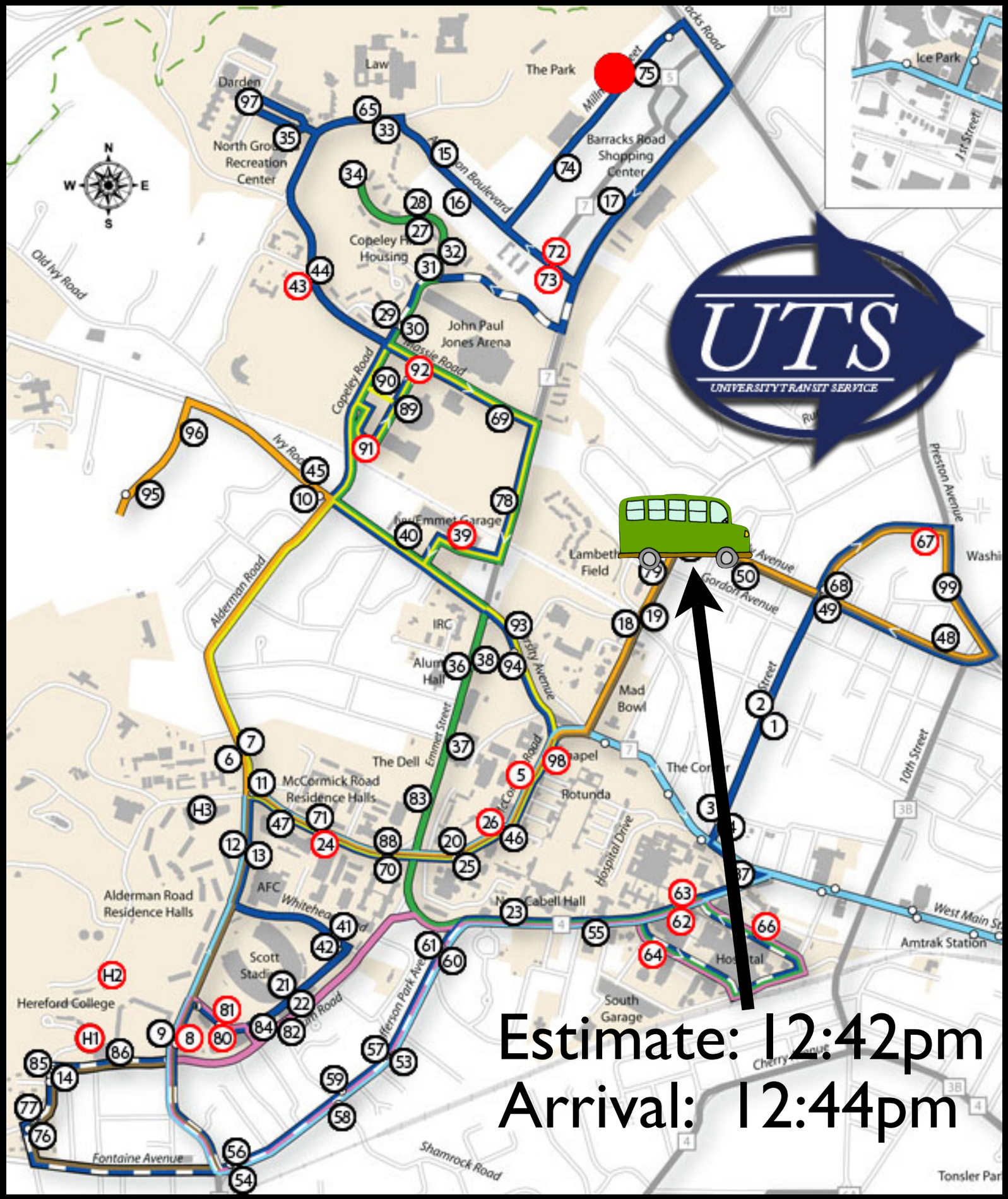
Almost no overhead

Application	Execution
Surge	17.7 msec
MacroLab_Surge	18.2 msec

Almost no overhead

Application	Execution	Stack
Surge	17.7 msec	120 bytes
MacroLab_Surge	18.2 msec	124 bytes


UTS
UNIVERSITY TRANSIT SERVICE
N
W
E
S
The Park
Ice Park
1st Street
Barracks Road Shopping Center
Darden
North Grounds Recreation Center
Law
Old Ivy Road
Copeley Hill Housing
John Paul Jones Arena
Massie Road
Copeley Road
Ivy Road
Emmet Garage
Lambeth Field
Alderman Road
IRC
Alumni Hall
Emmet Street
Gordon Avenue
Preston Avenue
Mad Bowl
The Dell
Rotunda
The Corner
10th Street
Hospital Drive
Alderman Road Residence Halls
AFC
Scott Stadium
Cabell Hall
Hospital
South Garage
Amtrak Station
West Main St
Jefferson Park Ave
Cherry Avenue
Shamrock Road
Fontaine Avenue
Hereford College
Tonsler Park
H1
H2
H3


UTS
UNIVERSITY TRANSIT SERVICE
Ice Park
1st Street
The Park
Barracks Road Shopping Center
Darden
North Grounds Recreation Center
Copeley Hill Housing
John Paul Jones Arena
Old Ivy Road
Copeley Road
Massie Road
Ivy Road
Alderman Road
Emmet Garage
Lambeth Field
Gordon Avenue
Preston Avenue
IRC
Alumni Hall
University Avenue
Emmet Street
Mad Bowl
The Dell
McCormick Road Residence Halls
Alderman Road Residence Halls
AFC
Scott Stadium
Hereford College
Fontaine Avenue
Shamrock Road
Jefferson Park Avenue
Rotunda
Hospital Drive
Cabell Hall
South Garage
10th Street
West Main Street
Amtrak Station
Tonsler Park
Estimate: 12:42pm
Arrival: 12:44pm

Bus Tracking

```
RTS = RunTimeSystem();
busstops = RTS.getNodes('stopnode');
buses = RTS.getNodes('bus');
estimates = Macrovector(busstops, length(buses) ,'uint16');
arrivals = Macrovector(busstops, length(buses) ,'uint16'));
travelTime = Macrovector(busstops, length(busstops), length(buses) ,'uint16'));
busSensors = SensorVector('BusSensor',busstops,'uint16');
routes = uint8({[1 2 3 4], [ 5 6 7 8]}); %Example routes

every(1000)
  [busID,r] =  busSensors.sense();
  busTime = RTS.getTime();
  travelTime(routes{r},routes{r},busID)[1,3] = busTime - arrivals(routes{r}, busID);
  arrivals(routes{r},busID)[1,2] = busTime;
  estimates(routes{r},busID) = travelTime(routes{r},routes{r},busID)[2,3] + busTime;
  baseDisplay(estimates(routes{r},:));
end
```

Bus Tracking

```
RTS = RunTimeSystem();
busstops = RTS.getNodes('stopnode');
buses = RTS.getNodes('bus');
estimates = Macrovector(busstops, length(buses) ,'uint16');
arrivals = Macrovector(busstops, length(buses) ,'uint16'));
travelTime = Macrovector(busstops, length(busstops), length(buses) ,'uint16'));
busSensors = SensorVector('BusSensor',busstops,'uint16');
routes = uint8({[1 2 3 4], [ 5 6 7 8]}); %Example routes

every(1000)
  [busID,r] =  busSensors.sense();
  busTime = RTS.getTime();
  travelTime(routes{r},routes{r},busID)[1,3] = busTime - arrivals(routes{r}, busID);
  arrivals(routes{r},busID)[1,2] = busTime;
  estimates(routes{r},busID) = travelTime(routes{r},routes{r},busID)[2,3] + busTime;
  baseDisplay(estimates(routes{r},:));
end
```

Bus Tracking

```
RTS = RunTimeSystem();
busstops = RTS.getNodes('stopnode');
buses = RTS.getNodes('bus');
estimates = Macrovector(busstops, length(buses) ,'uint16');
arrivals = Macrovector(busstops, length(buses) ,'uint16'));
travelTime = Macrovector(busstops, length(busstops), length(buses) ,'uint16'));
busSensors = SensorVector('BusSensor',busstops,'uint16');
routes = uint8({[1 2 3 4], [ 5 6 7 8]}); %Example routes
```

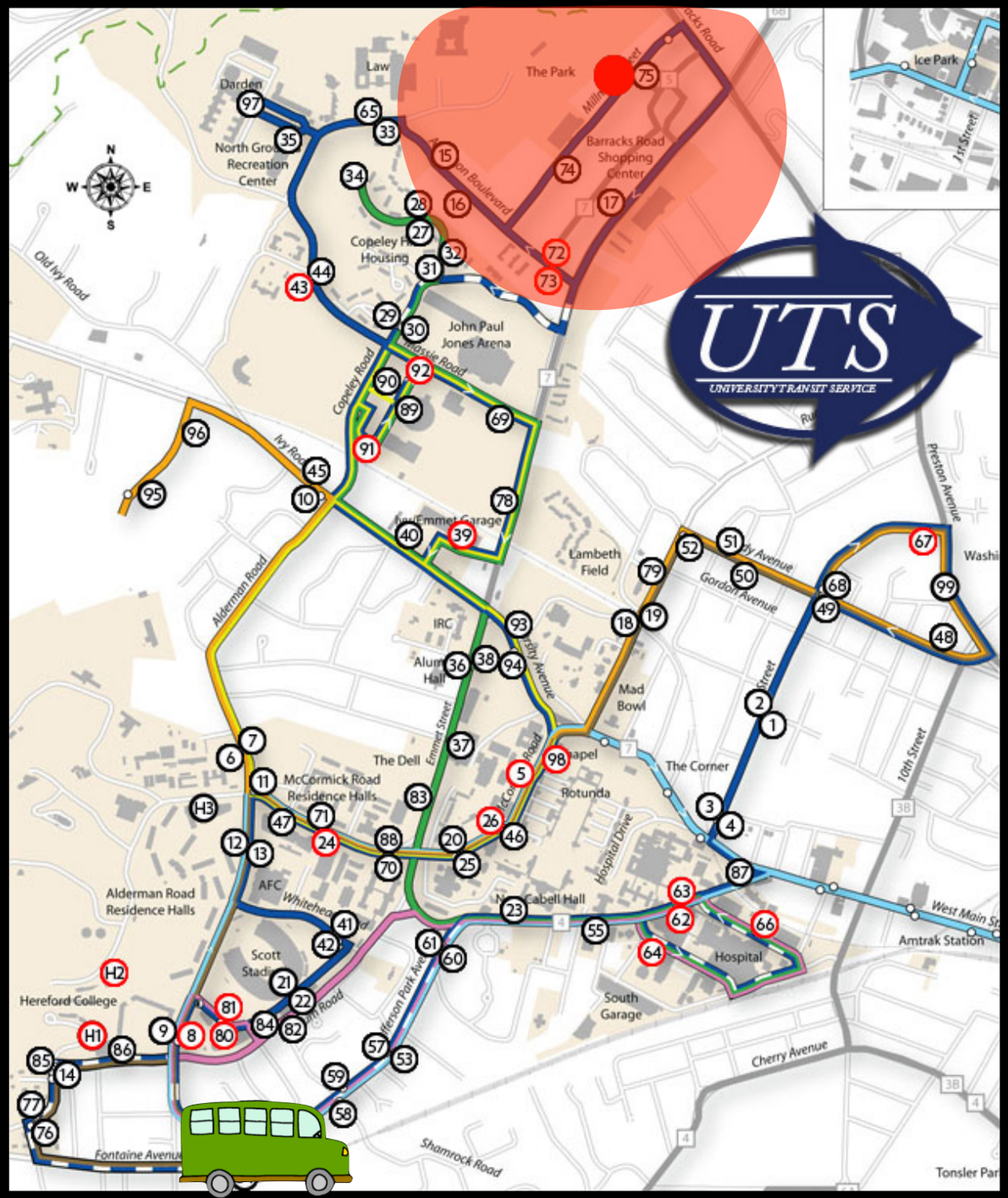
```
every(1000)
  [busID,r] =  busSensors.sense();
  busTime = RTS.getTime();
  travelTime(routes{r},routes{r},busID)[1,3] = busTime - arrivals(routes{r}, busID);
  arrivals(routes{r},busID)[1,2] = busTime;
  estimates(routes{r},busID) = travelTime(routes{r},routes{r},busID)[2,3] + busTime;
  baseDisplay(estimates(routes{r},:));
end
```

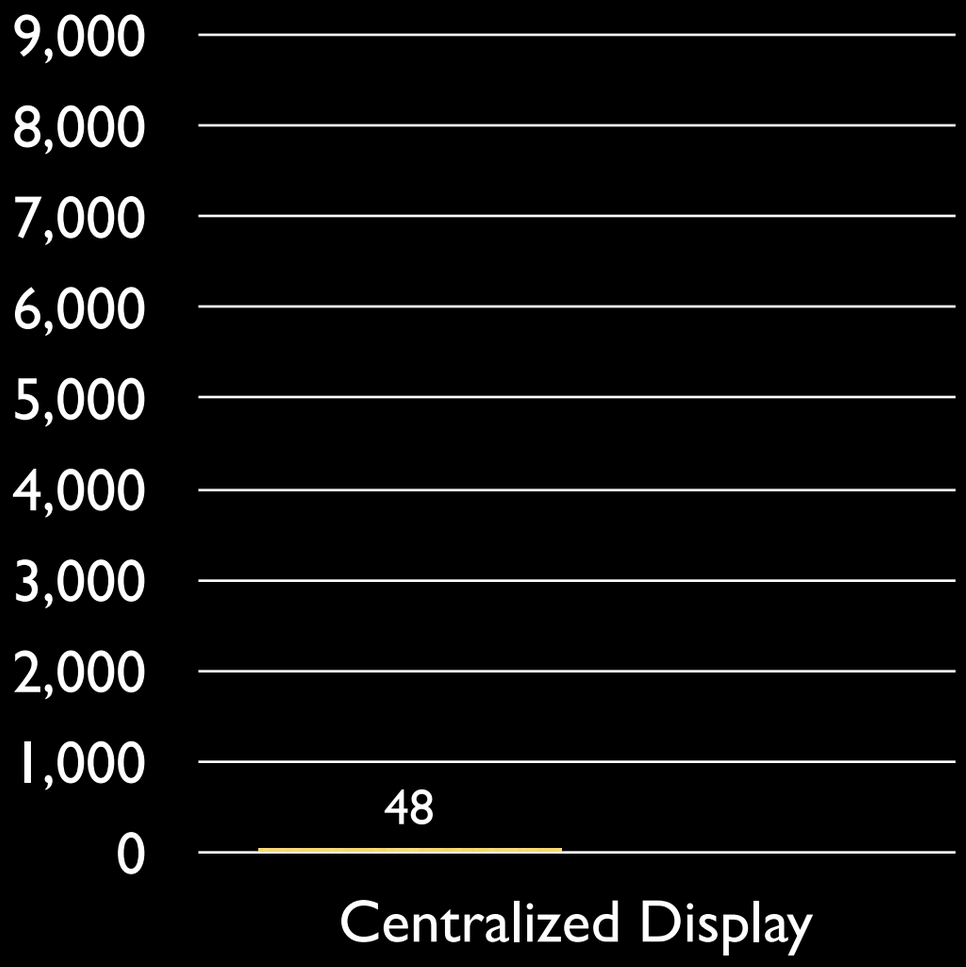
Bus Tracking

```
RTS = RunTimeSystem();
busstops = RTS.getNodes('stopnode');
buses = RTS.getNodes('bus');
estimates = Macrovector(busstops, length(buses) ,'uint16');
arrivals = Macrovector(busstops, length(buses) ,'uint16'));
travelTime = Macrovector(busstops, length(busstops), length(buses) ,'uint16'));
busSensors = SensorVector('BusSensor',busstops,'uint16');
routes = uint8({[1 2 3 4], [ 5 6 7 8]}); %Example routes
```

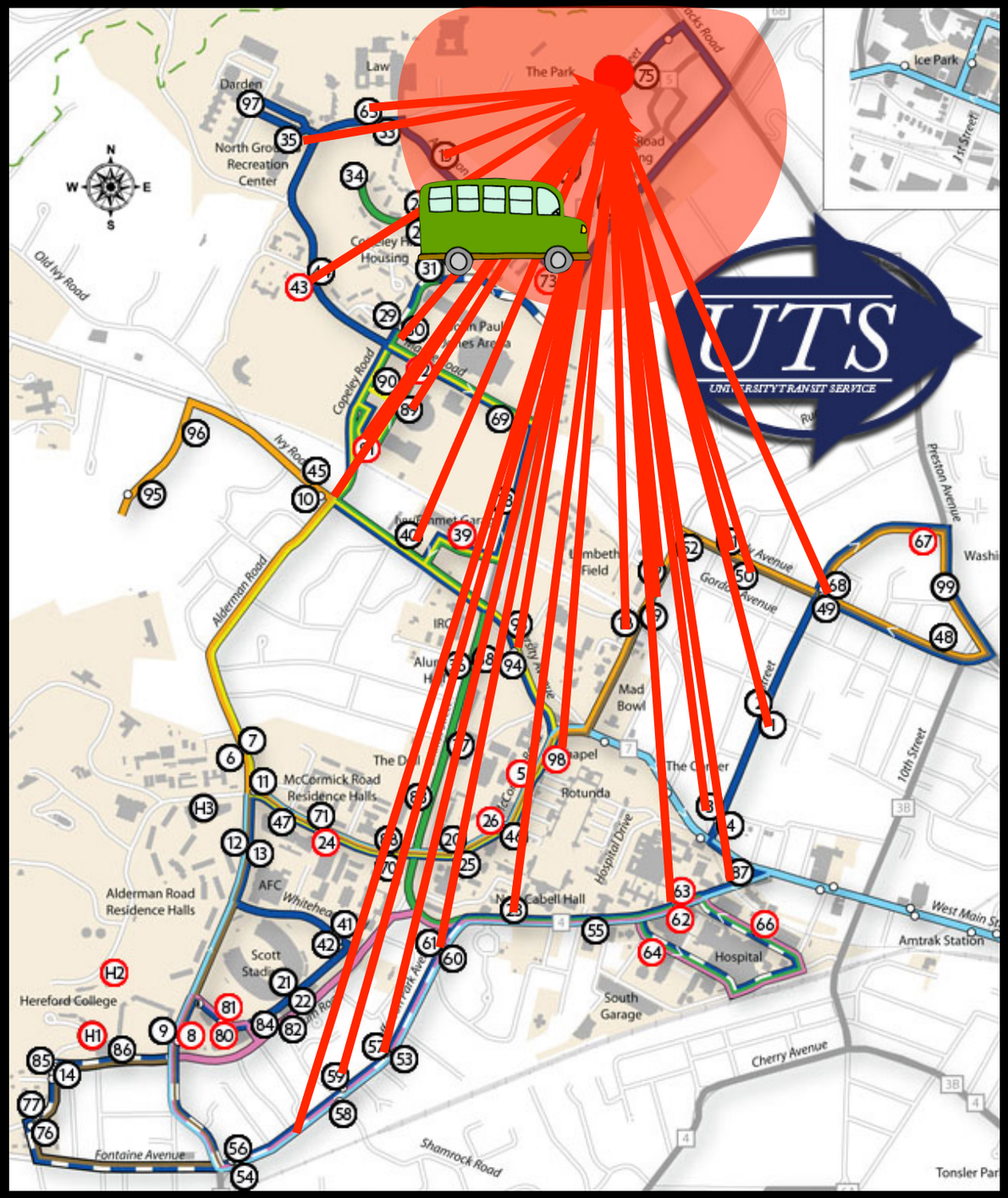
```
every(1000)
  [busID,r] =  busSensors.sense();
  busTime = RTS.getTime();
  travelTime(routes{r},routes{r},busID)[1,3] = busTime - arrivals(routes{r}, busID);
  arrivals(routes{r},busID)[1,2] = busTime;
  estimates(routes{r},busID) = travelTime(routes{r},routes{r},busID)[2,3] + busTime;
  baseDisplay(estimates(routes{r},:));
end
```

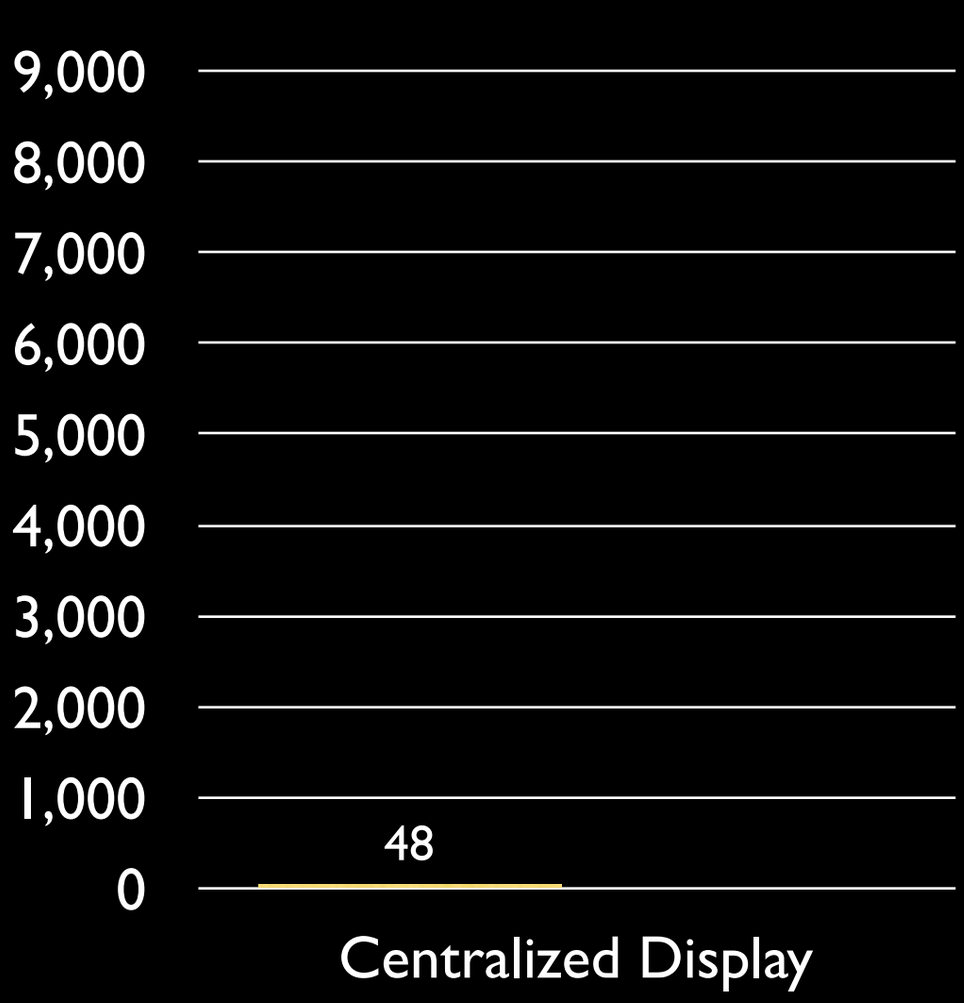
Simulation Results



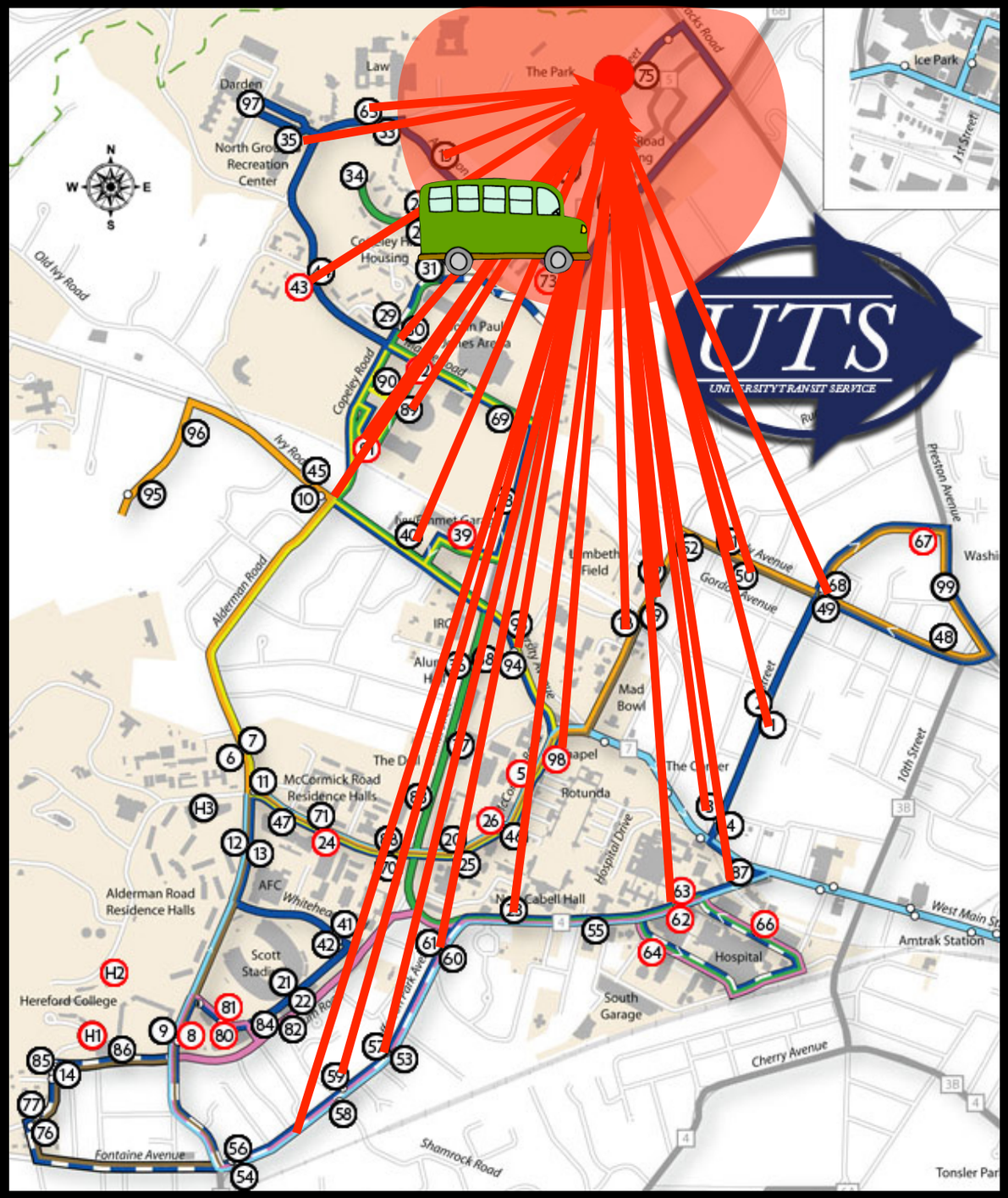


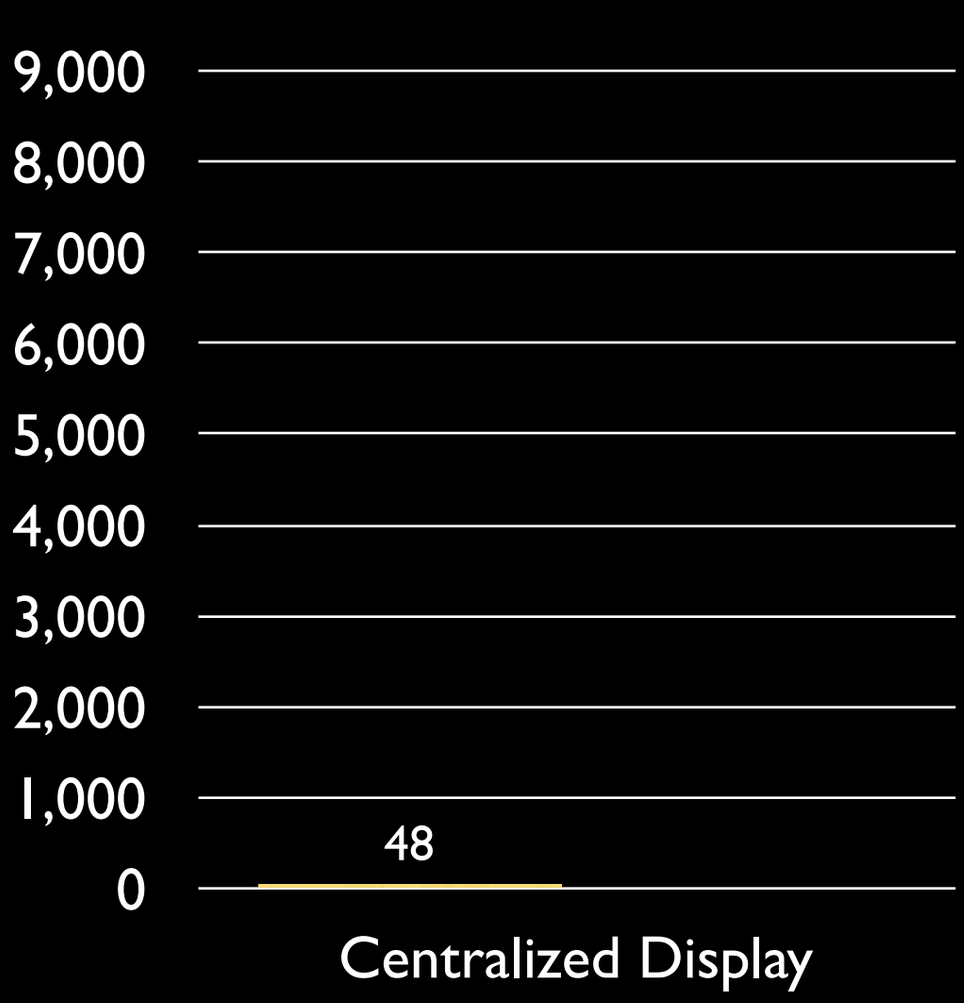
Simulation Results



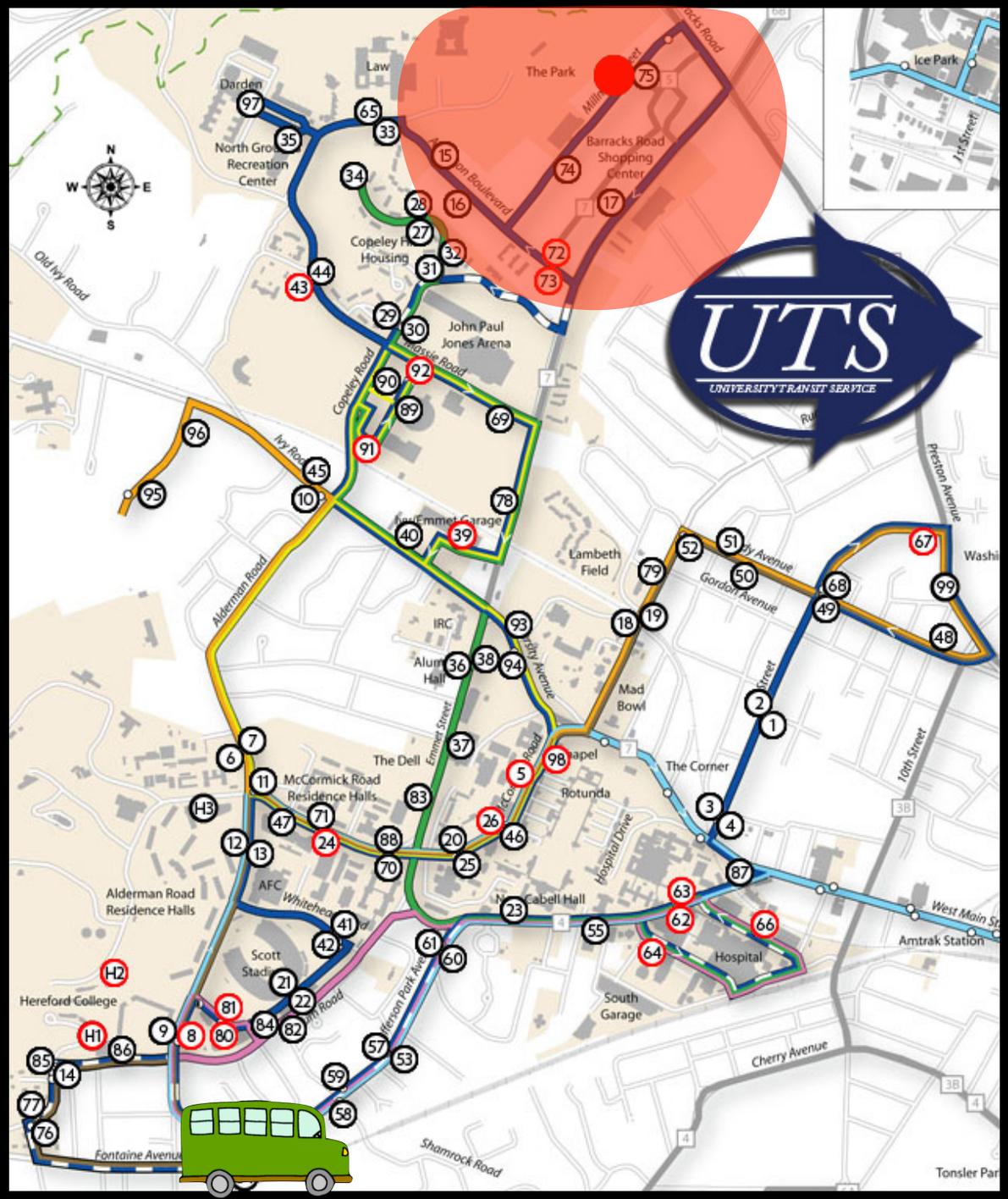


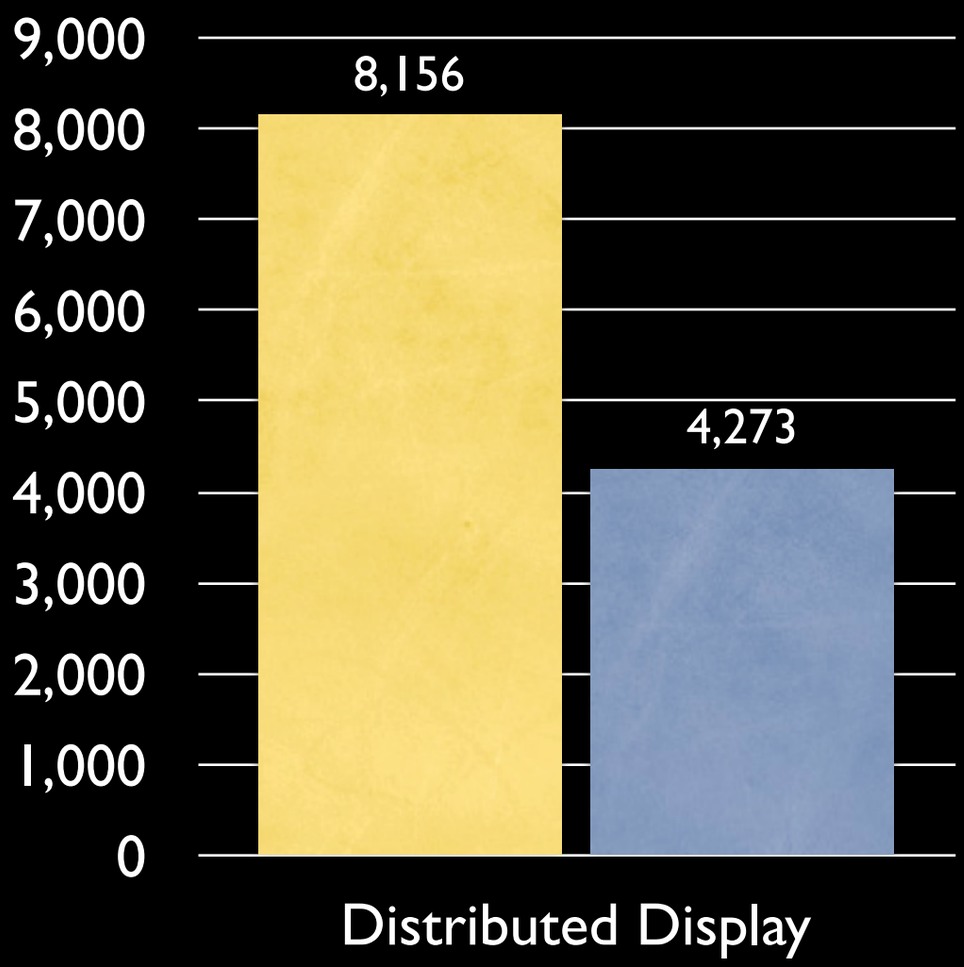
Simulation Results



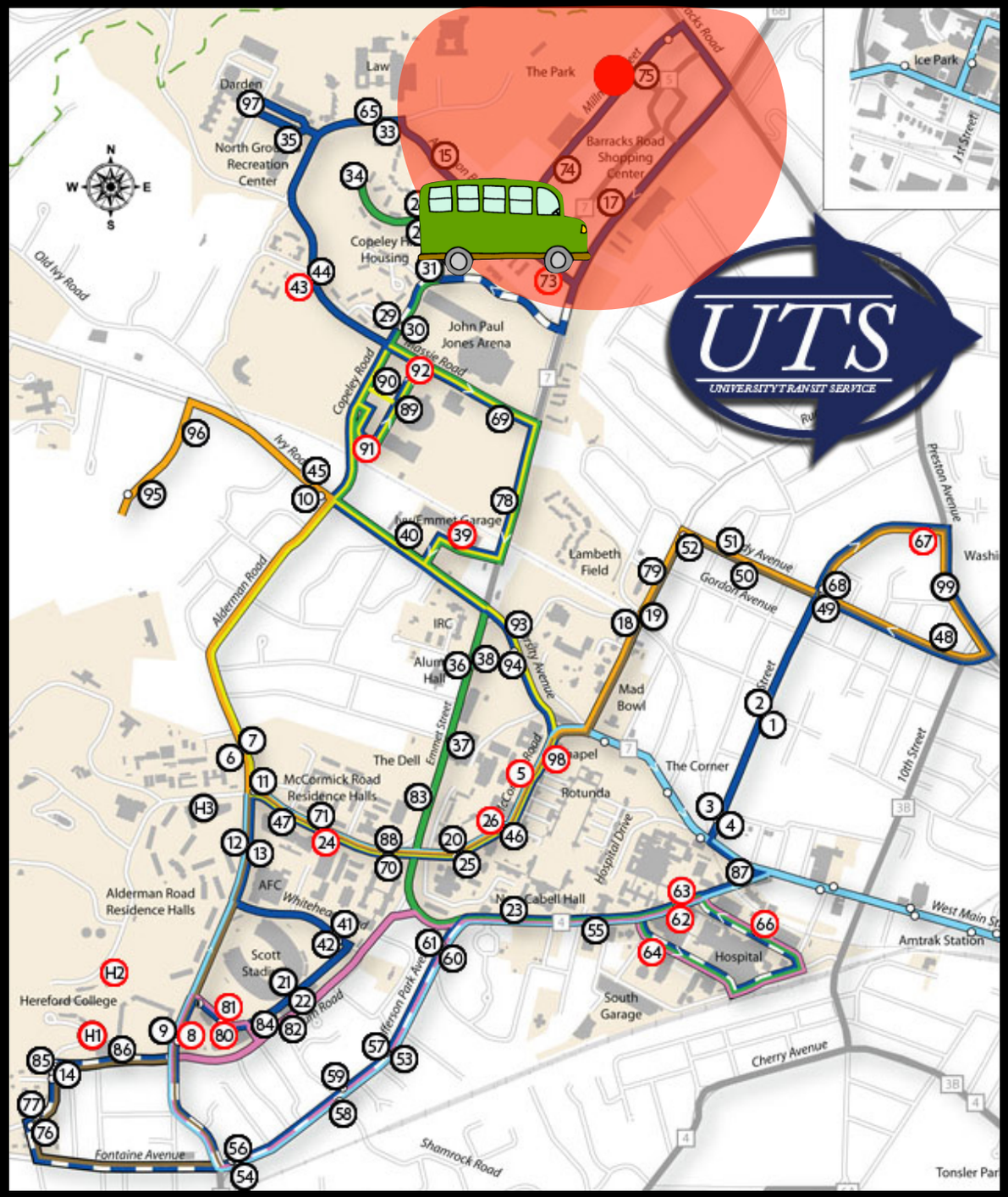


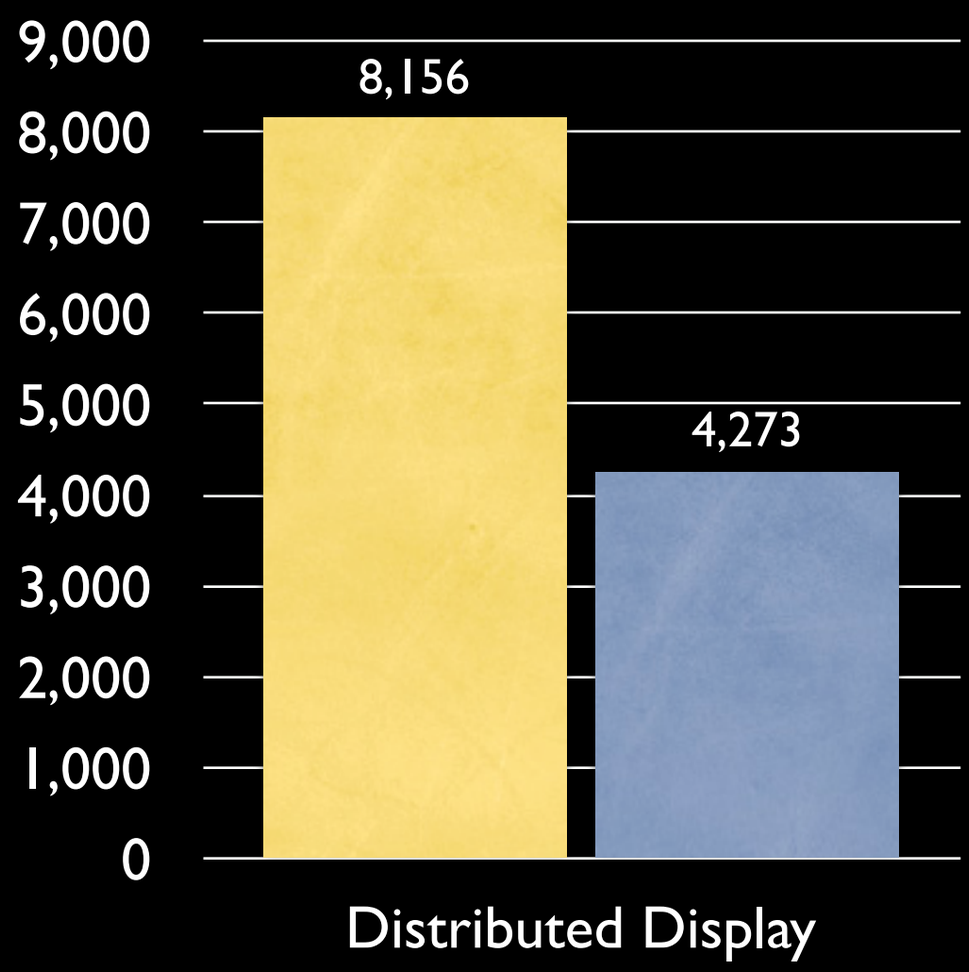
Simulation Results



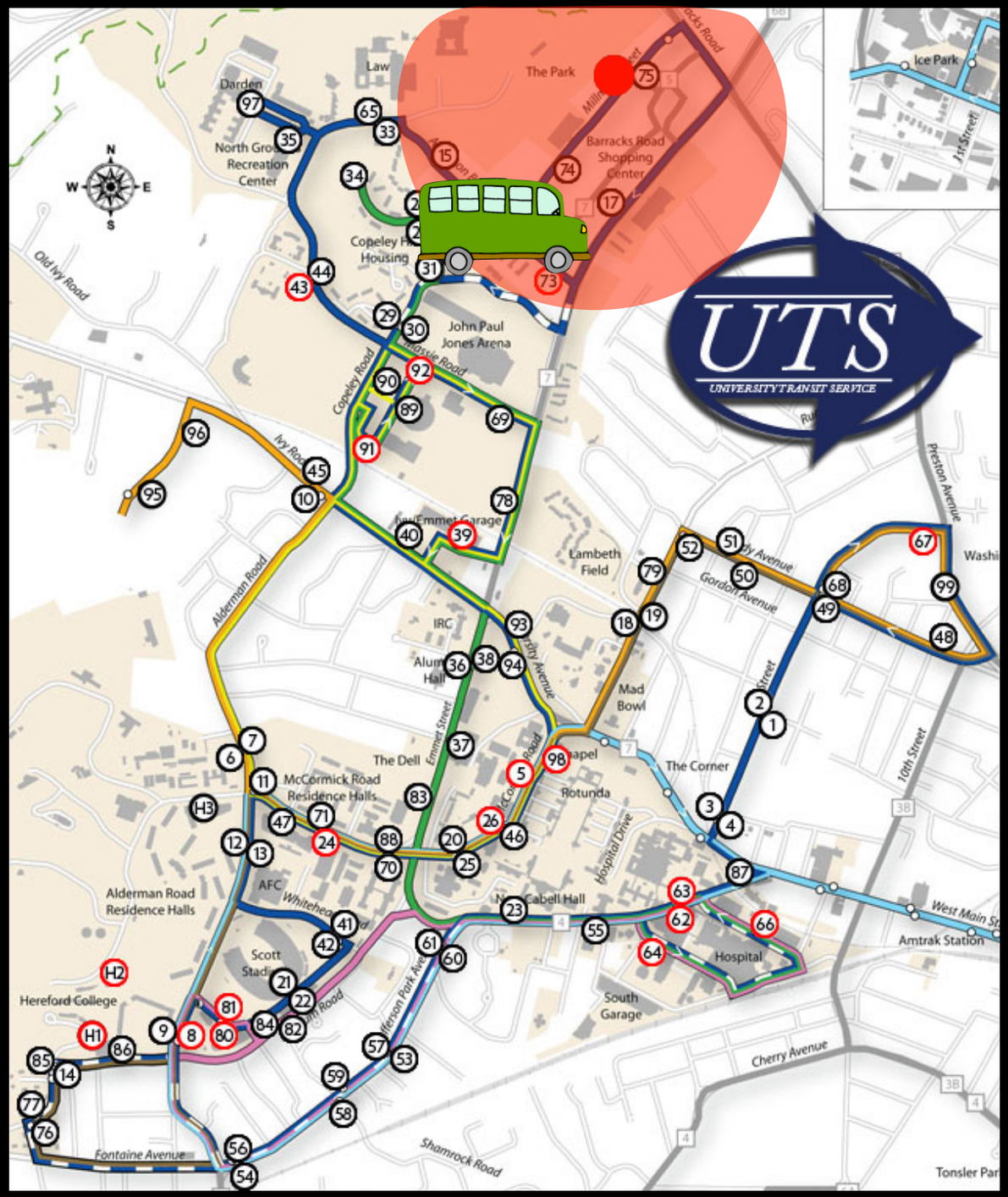


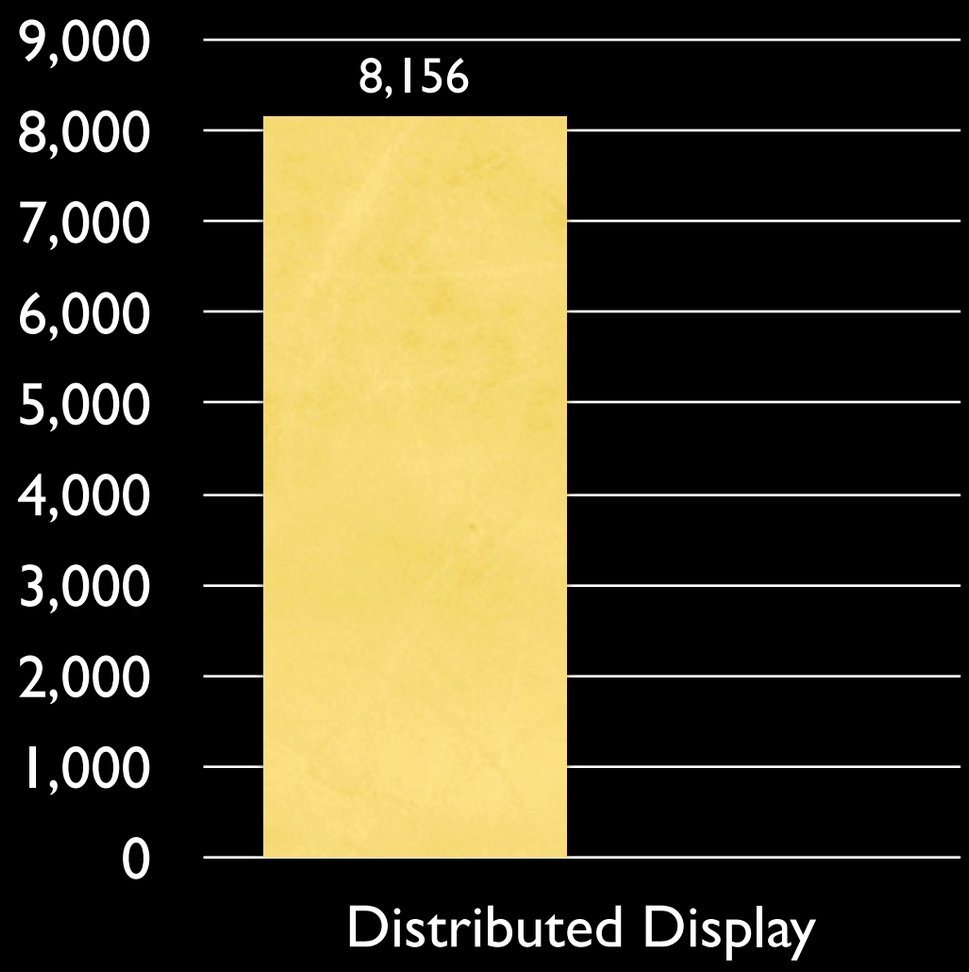
Simulation Results



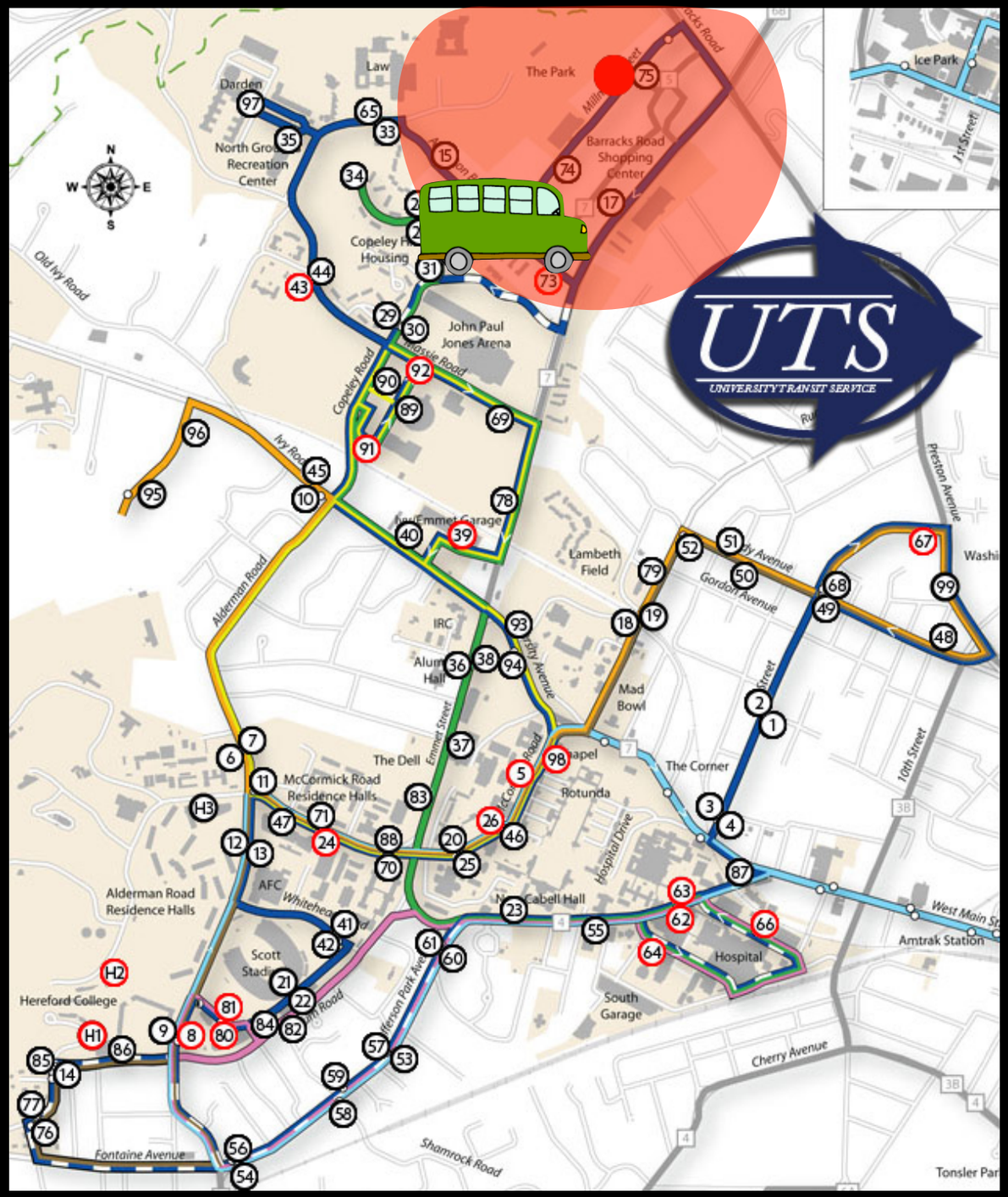


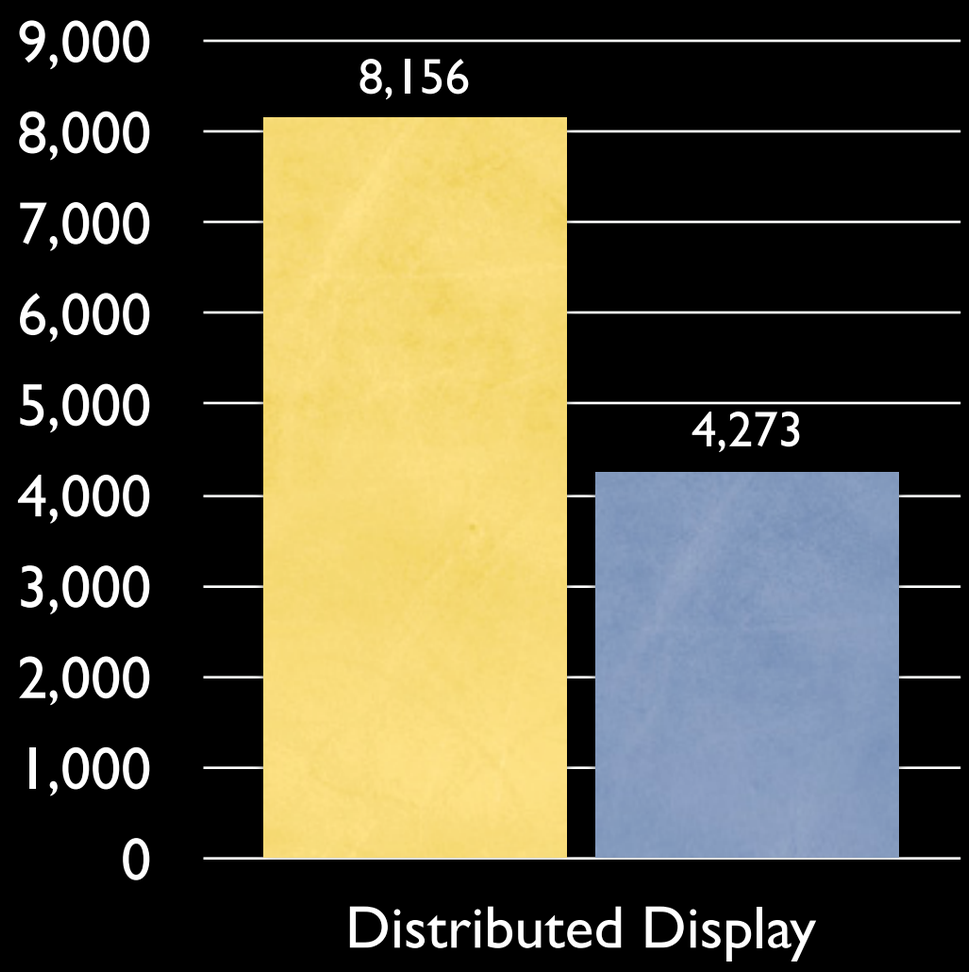
Simulation Results



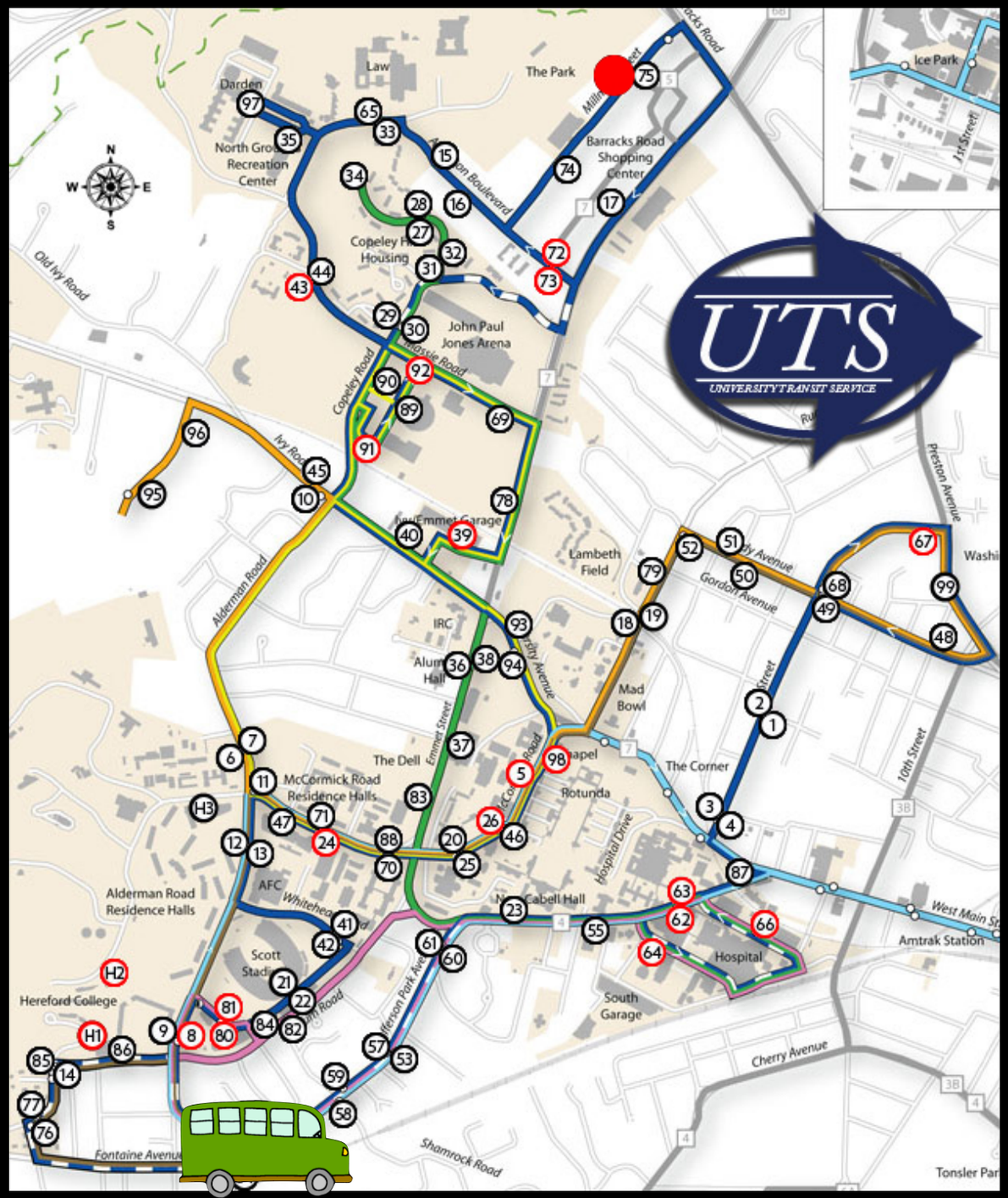


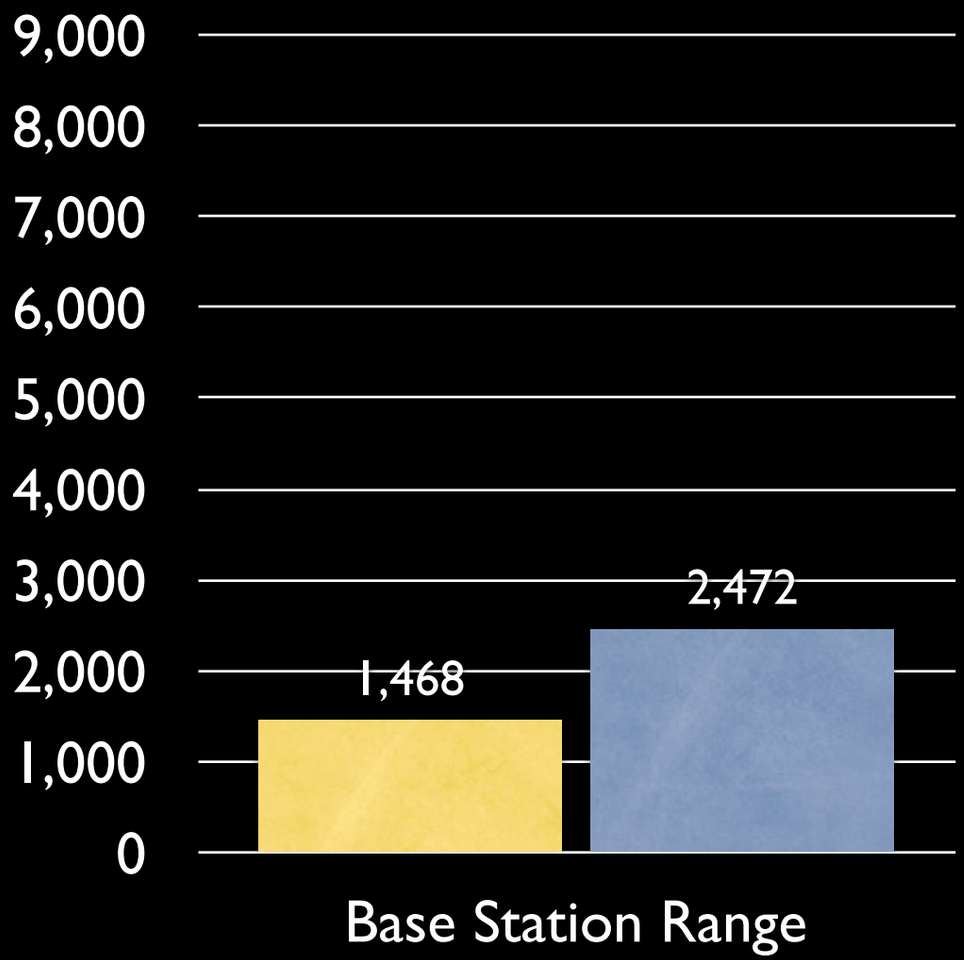
Simulation Results



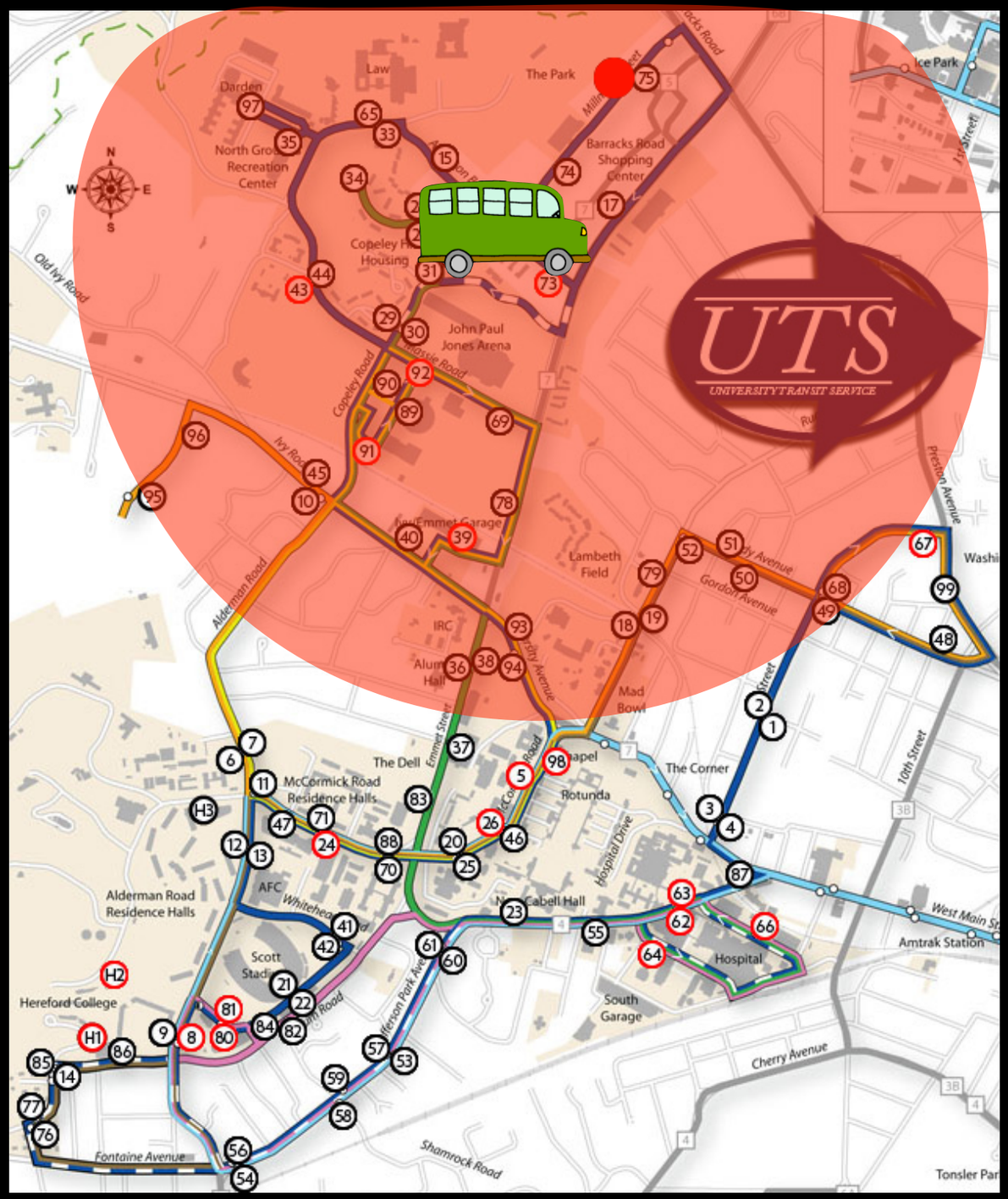


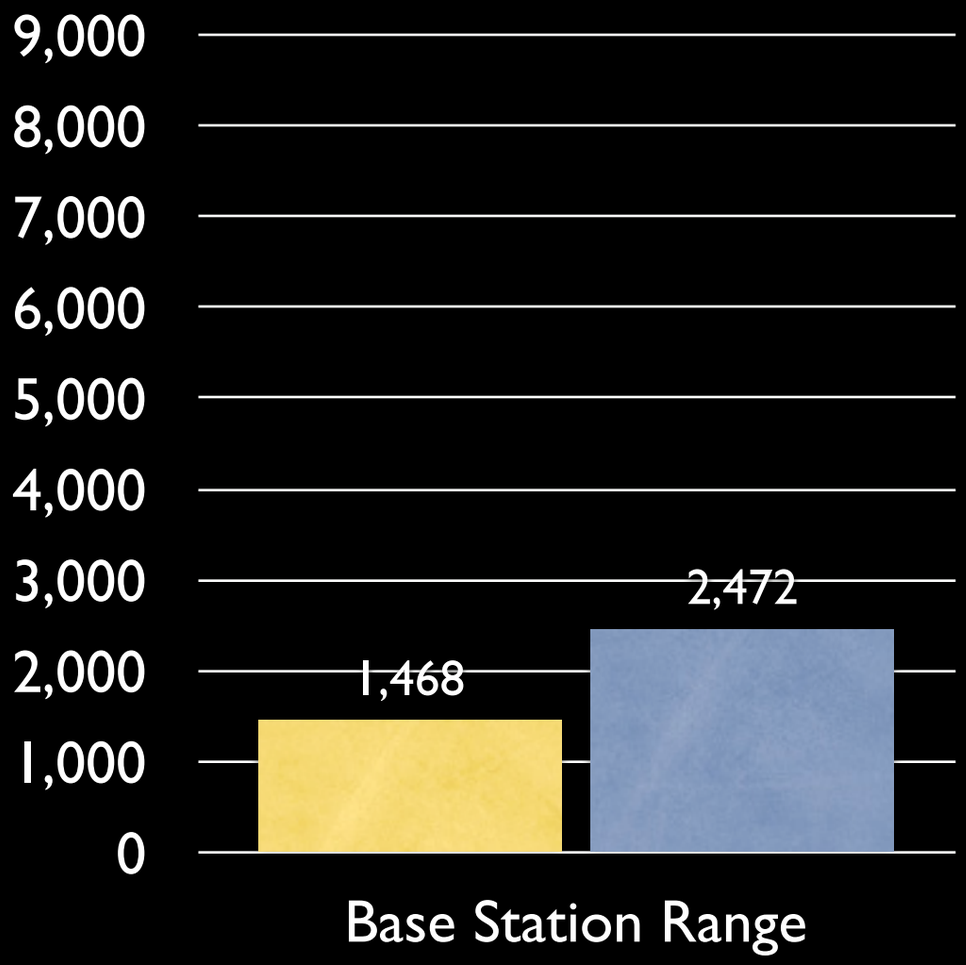
Simulation Results



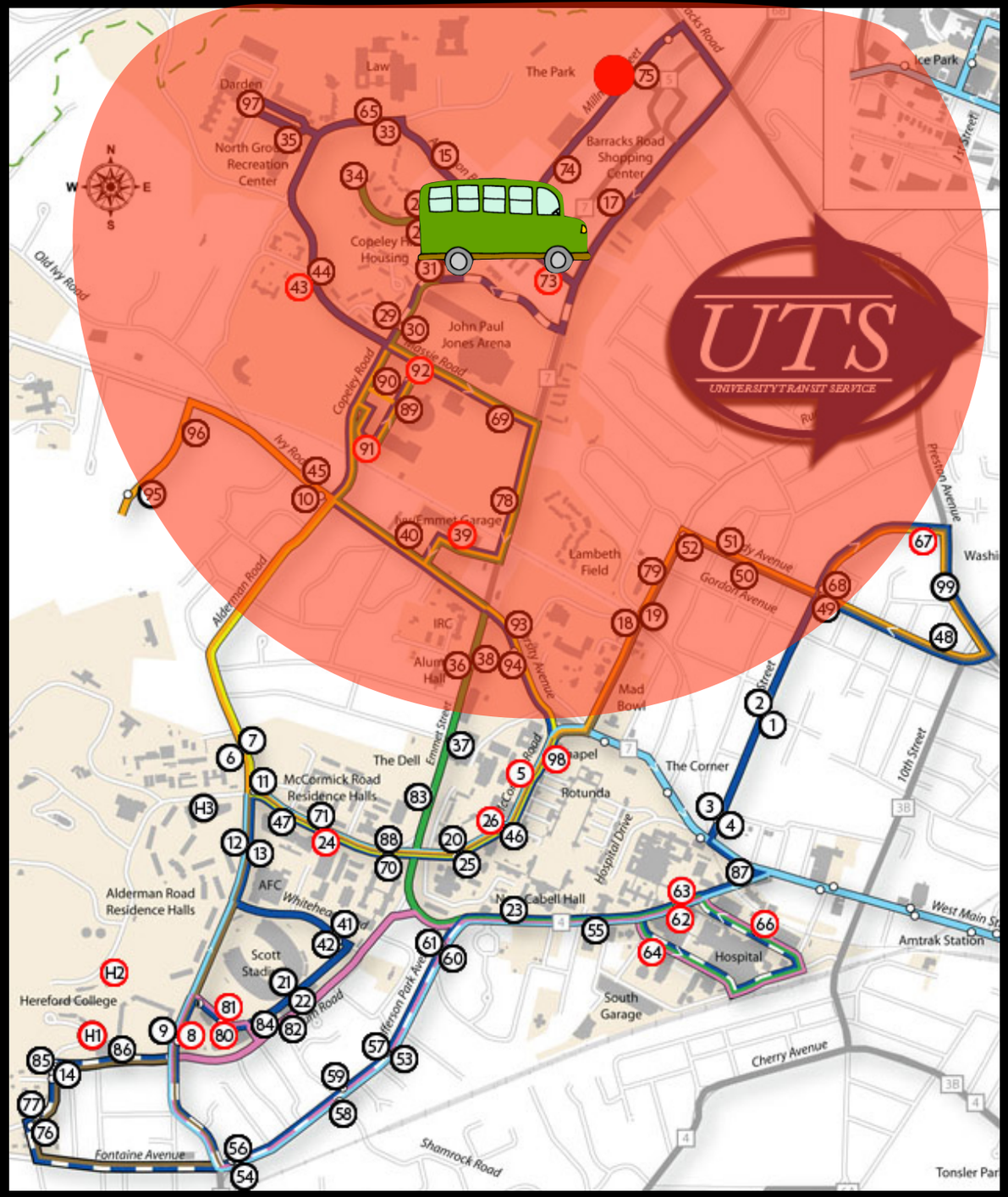


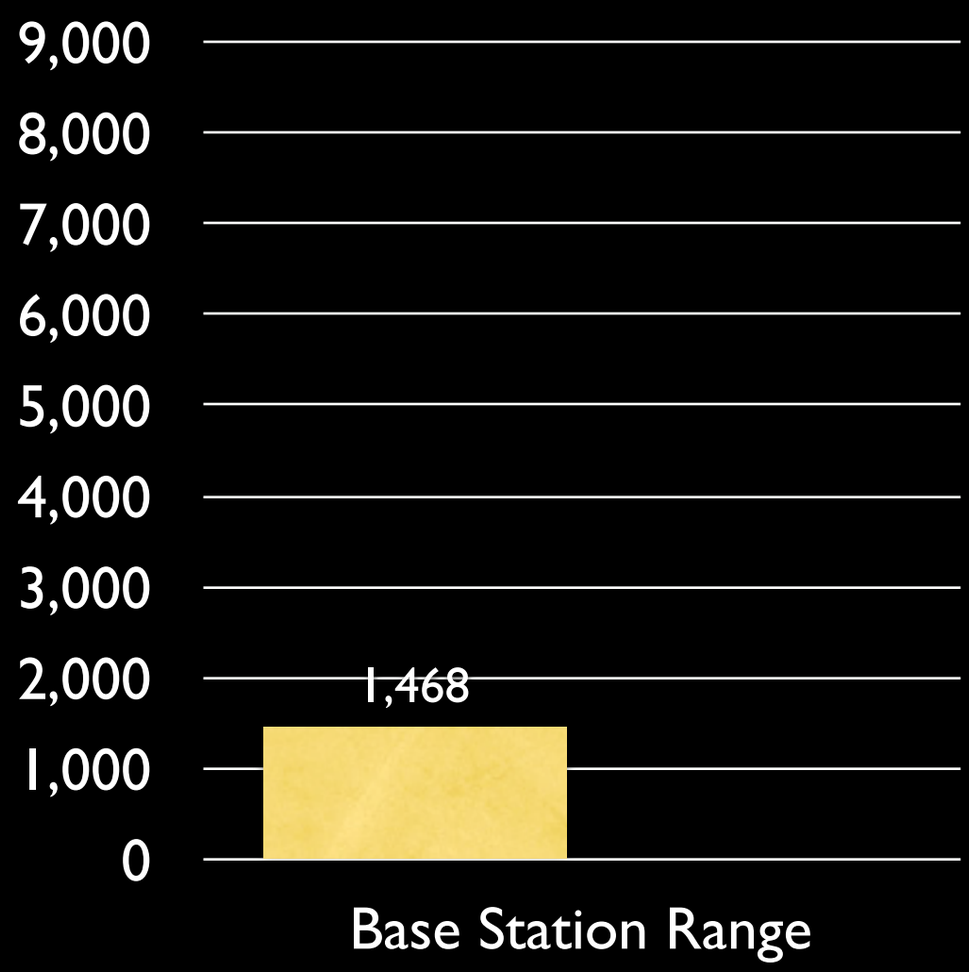
Simulation Results



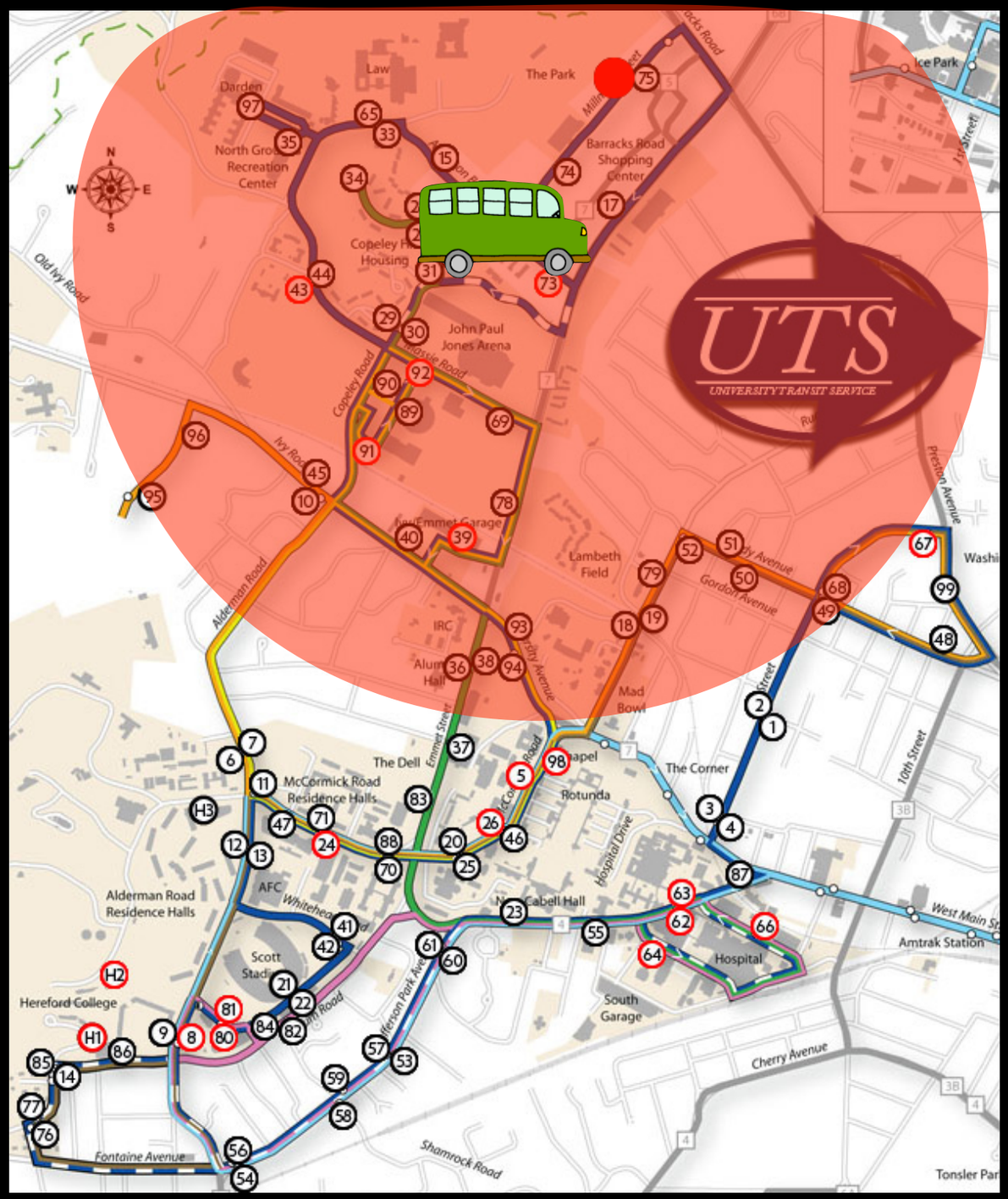


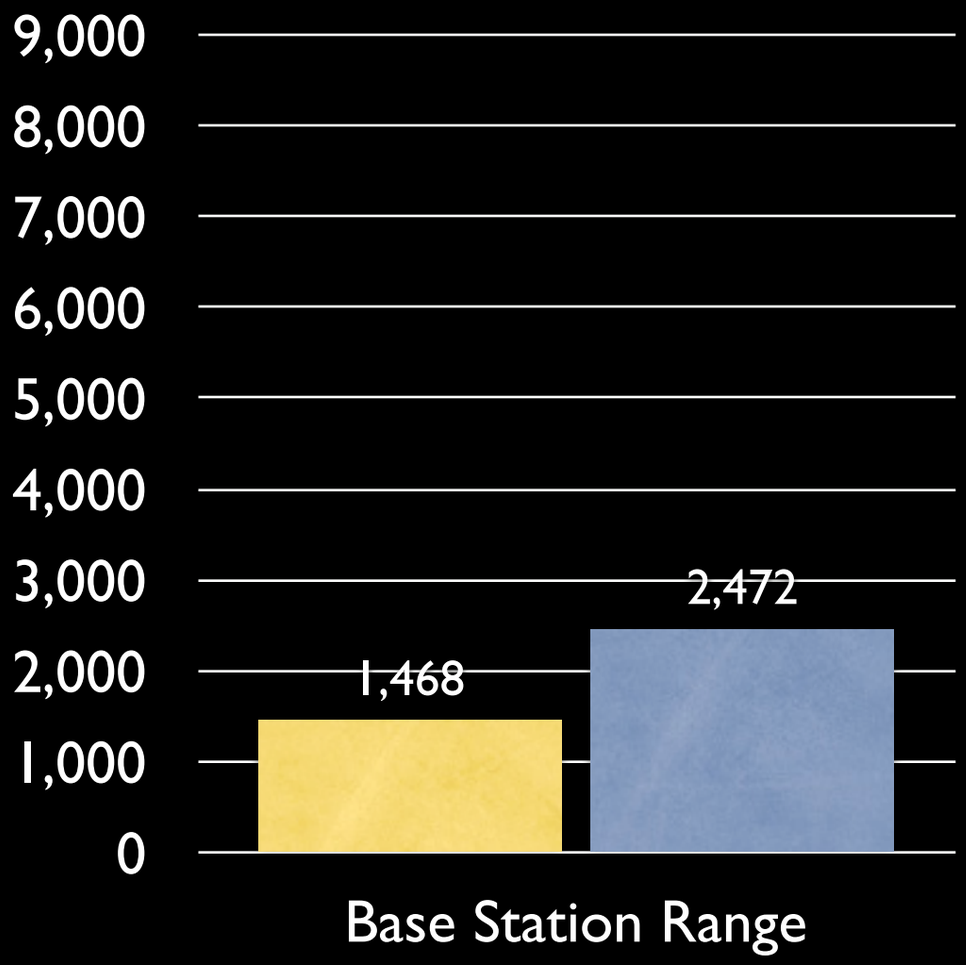
Simulation Results



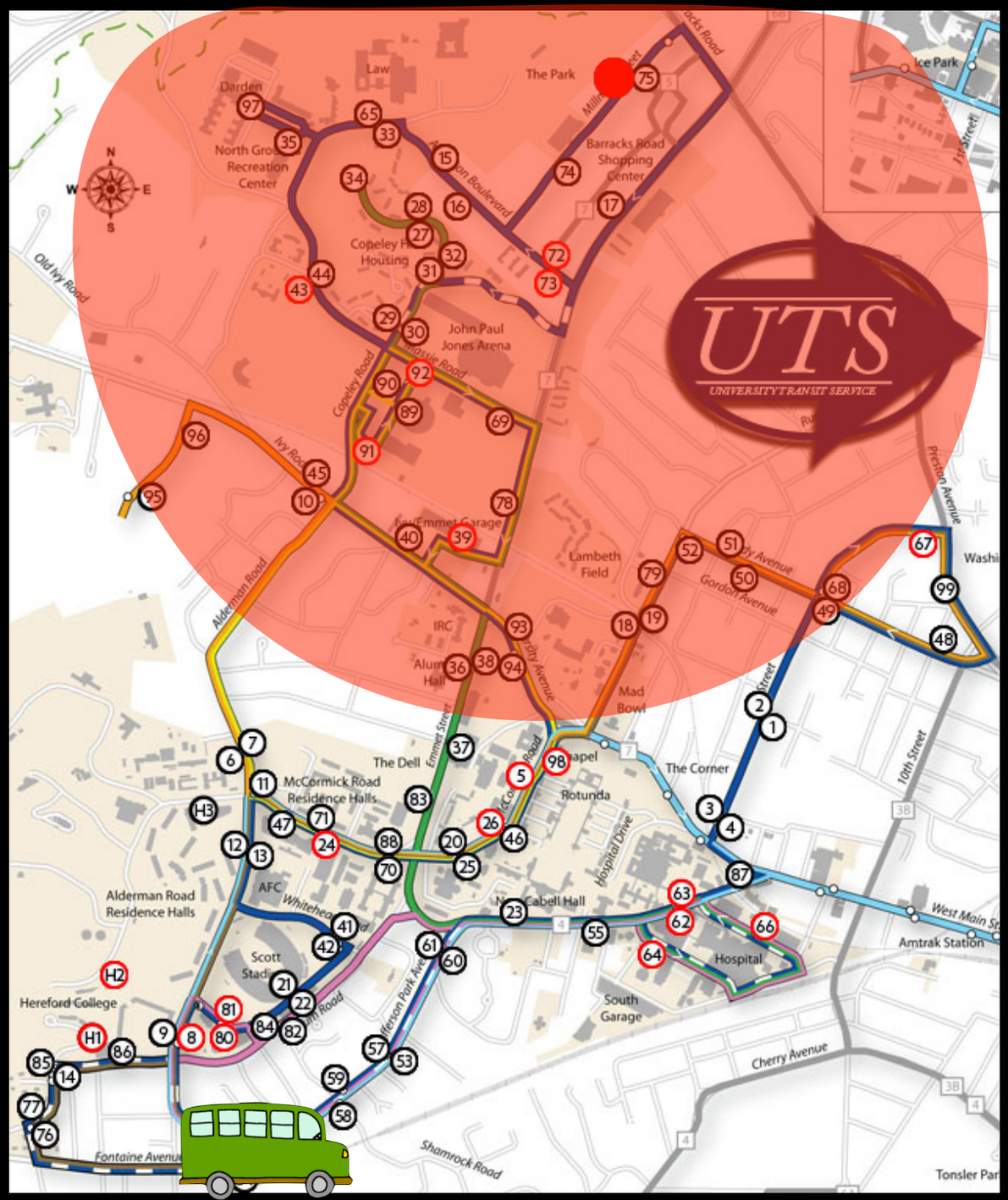


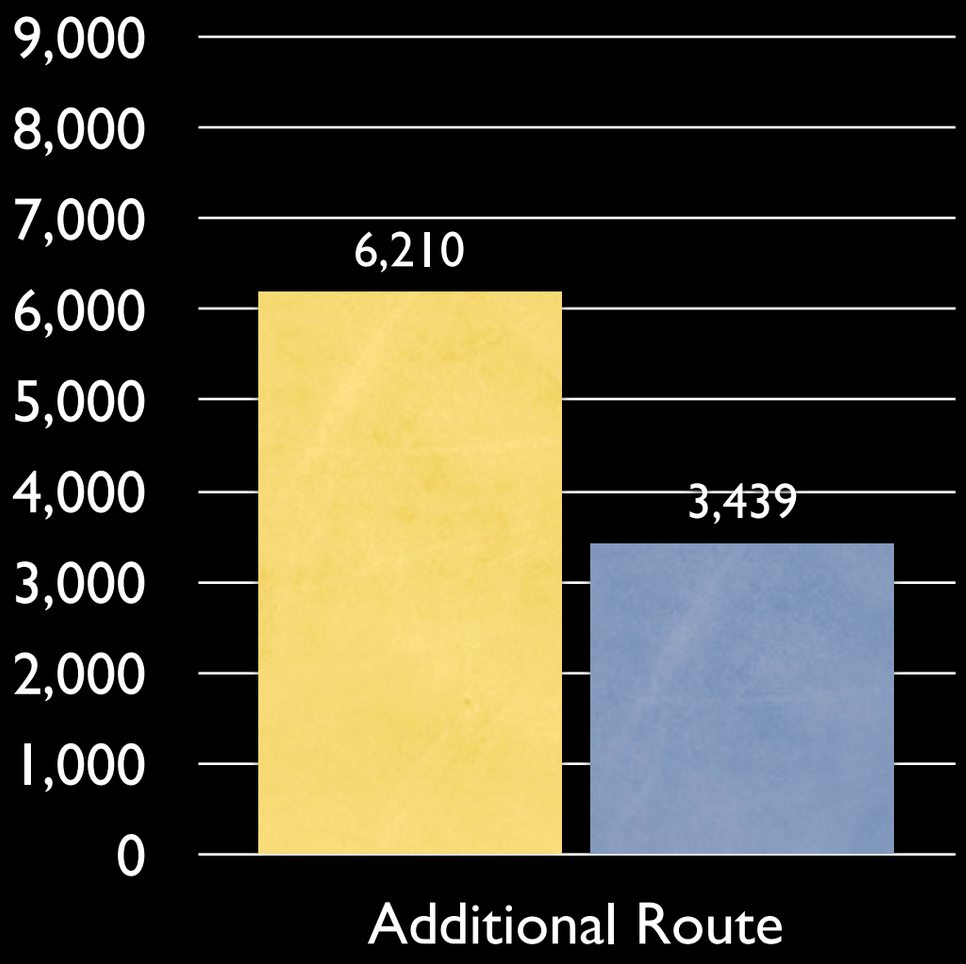
Simulation Results



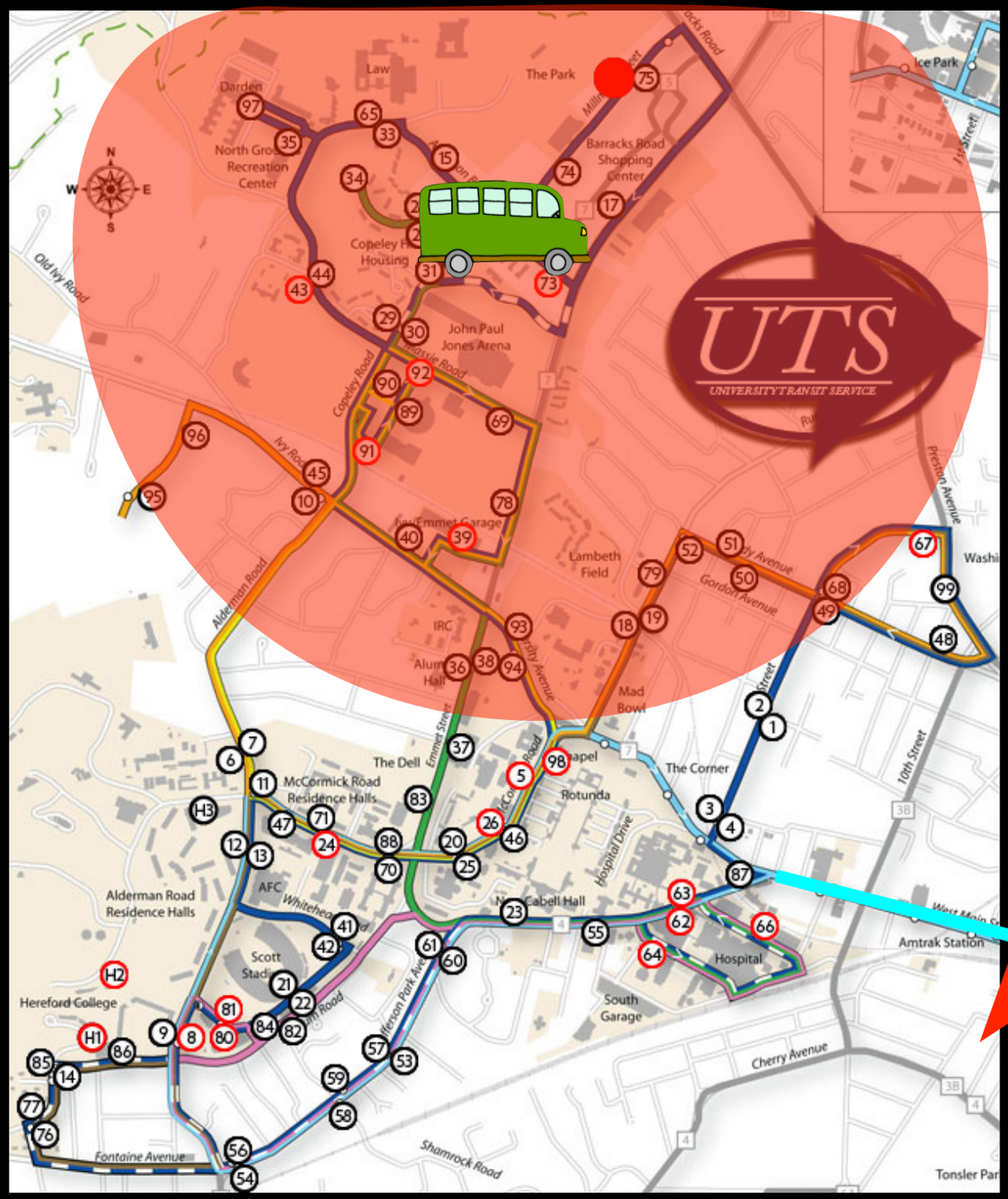


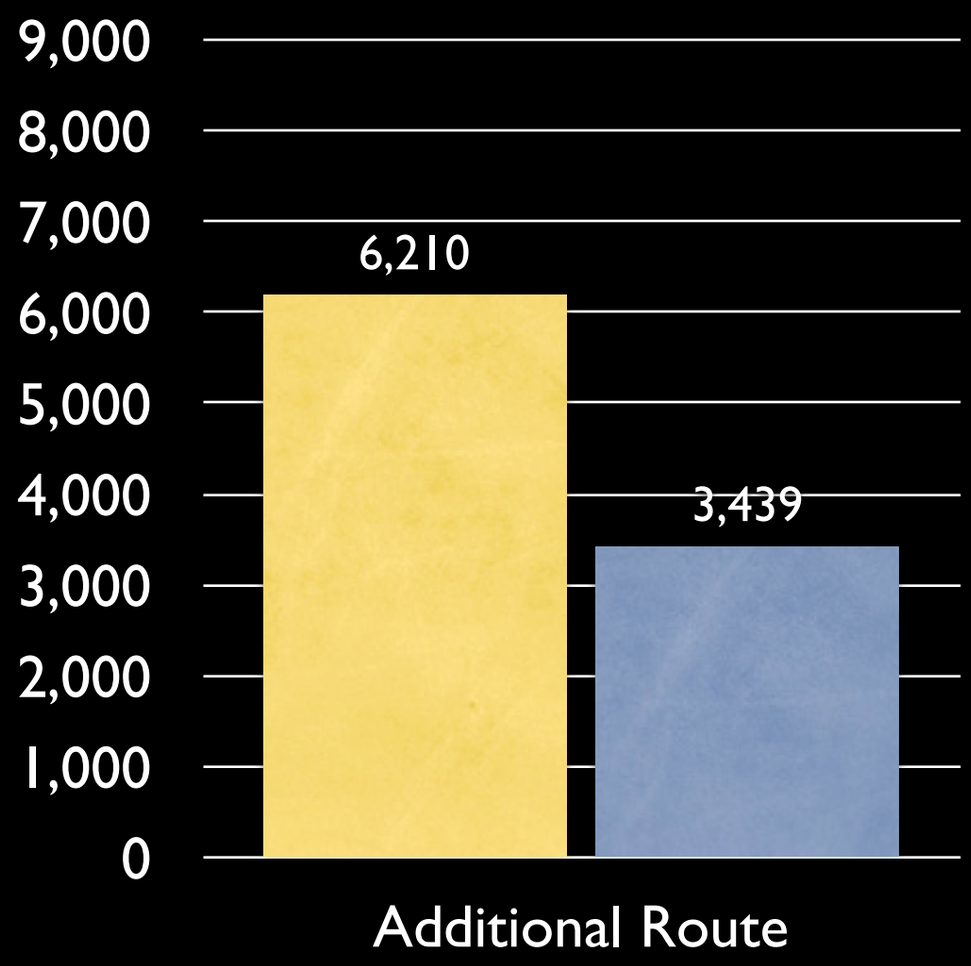
Simulation Results



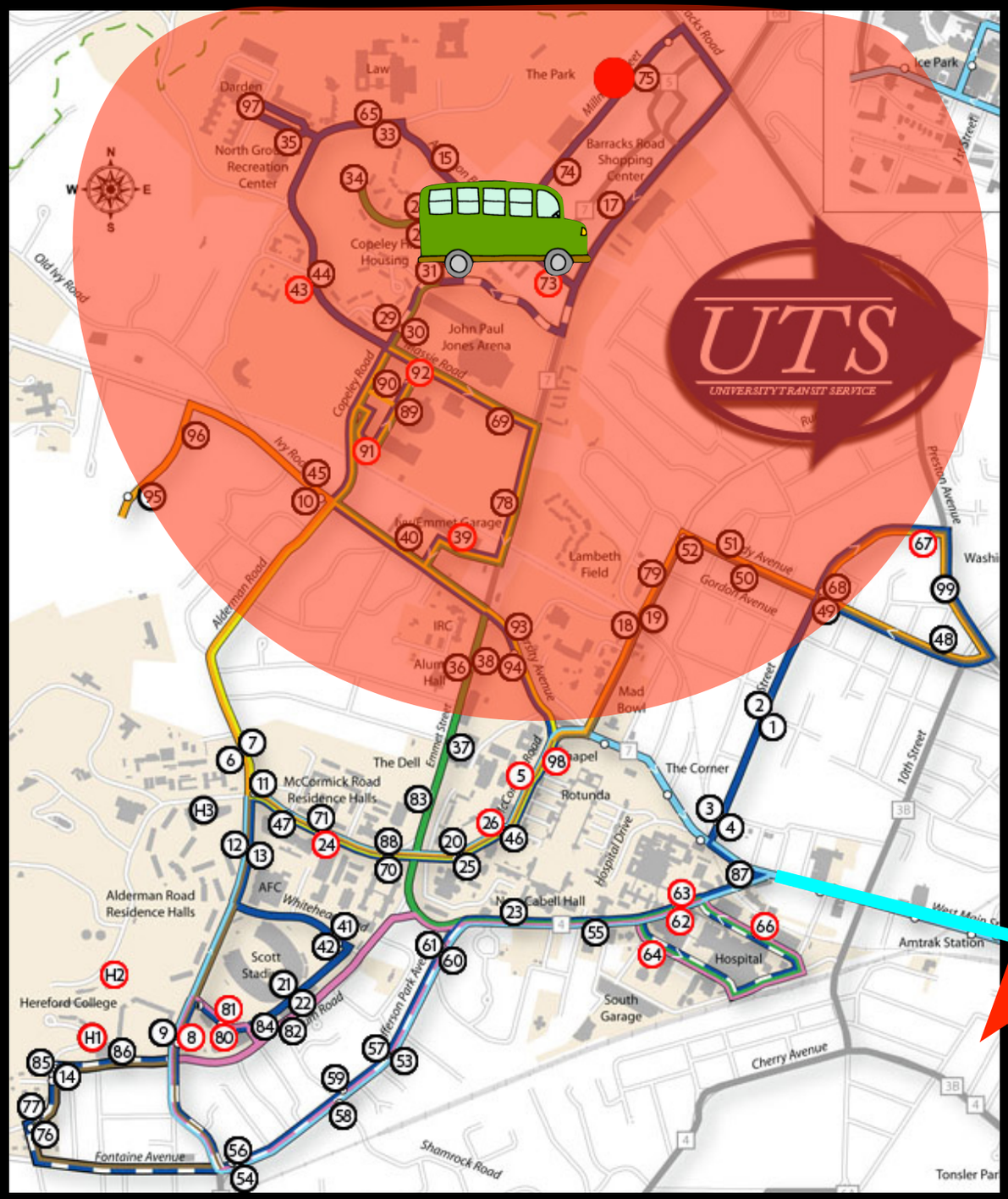


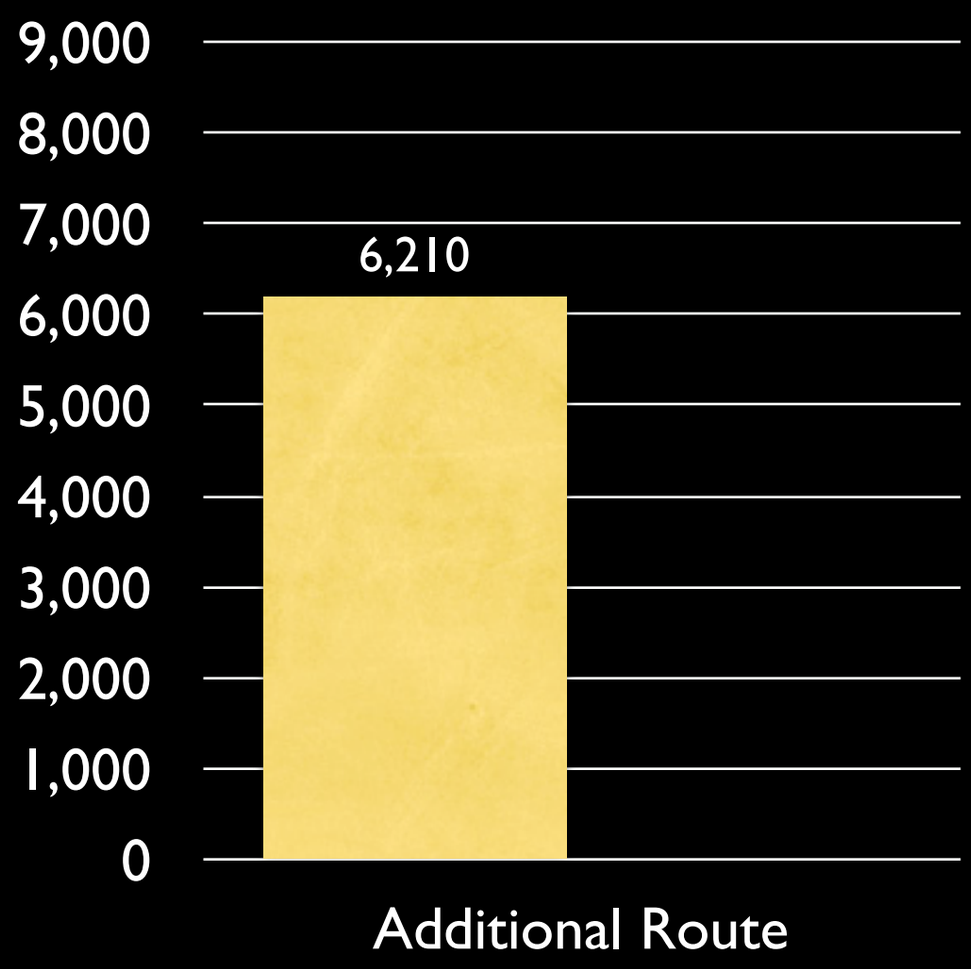
Simulation Results





Simulation Results

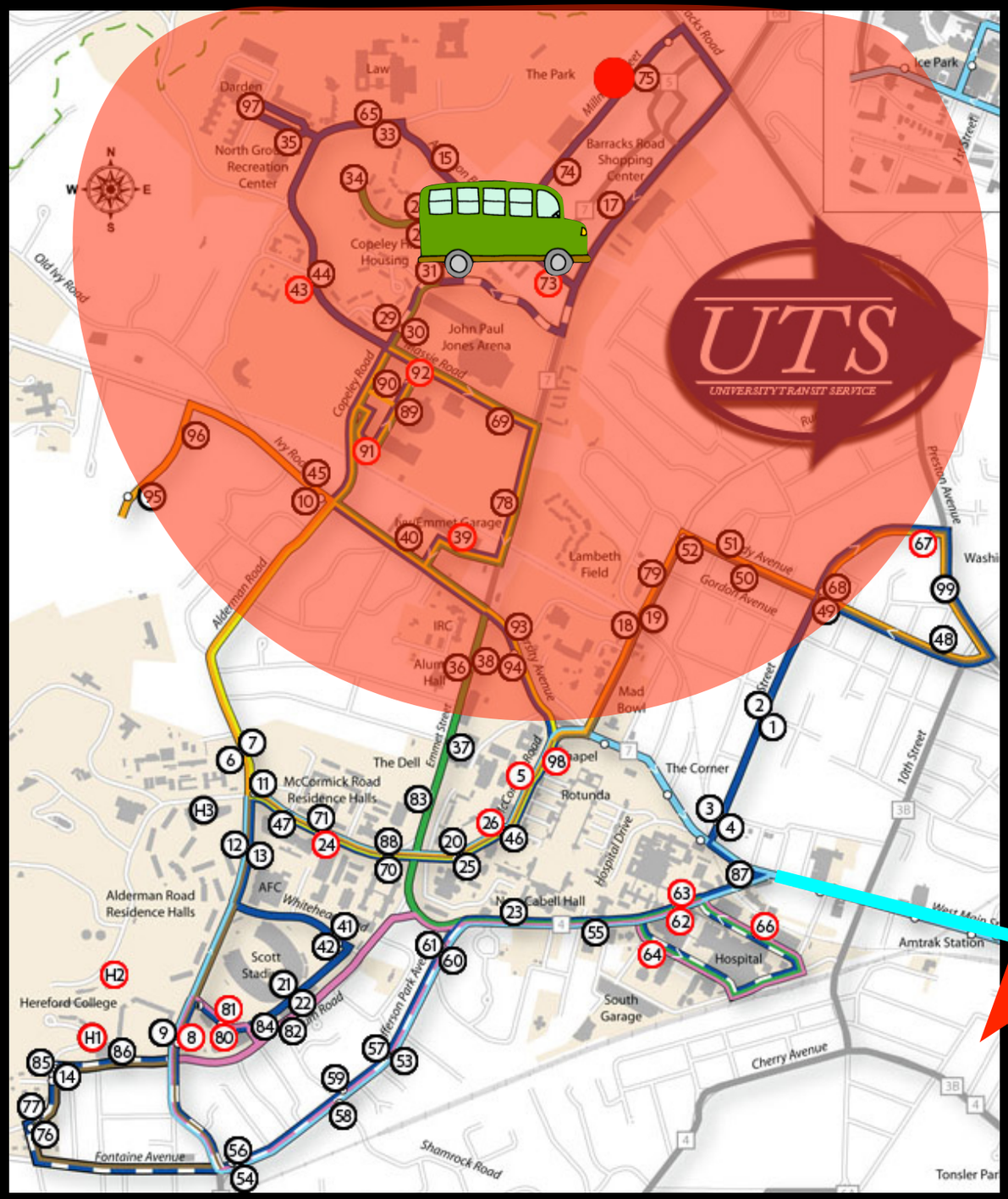


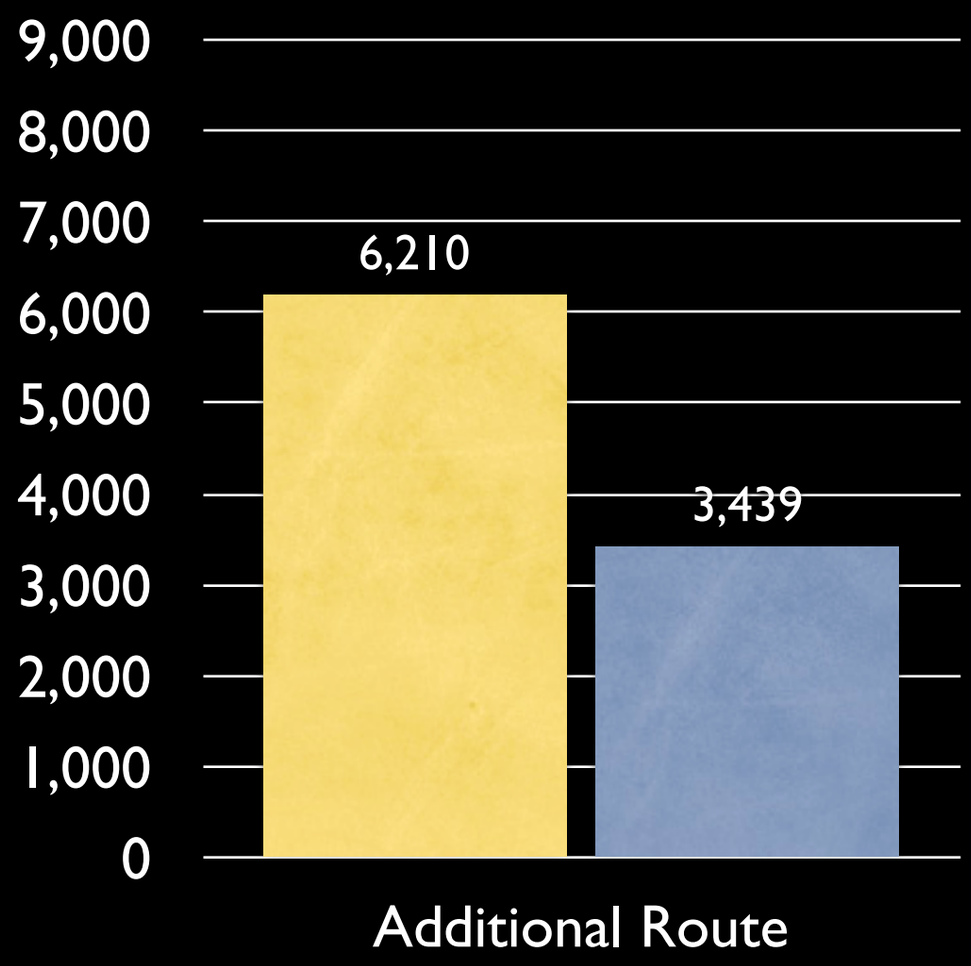


Centralized Implementation

Distributed Implementation

Simulation Results





Related Work

Related Work

- Node level programming
  - TinyOS, LiteOS, MantisOS, and Contiki

Related Work

- Node level programming
  - TinyOS, LiteOS, MantisOS, and Contiki
- Macroprogramming systems
  - TinyDB and Cougar: **Database**
  - Hood, Regions, and Proto: **Areas**
  - Semantic Streams, Flask, and Regiment: **Streams**

Related Work

- Node level programming
  - TinyOS, LiteOS, MantisOS, and Contiki
- Macroprogramming systems
  - TinyDB and Cougar: **Database**
  - Hood, Regions, and Proto: **Areas**
  - Semantic Streams, Flask, and Regiment: **Streams**
  - Marionette and Pleiades: **Imperative**

Related Work

Related Work

- User specified distribution
  - High Performance Fortran and Split-C

Related Work

- User specified distribution
  - High Performance Fortran and Split-C
- Automatic decomposition
  - MagnetOS, Coign, and J-Orchestra

Related Work

- User specified distribution
  - High Performance Fortran and Split-C
- Automatic decomposition
  - MagnetOS, Coign, and J-Orchestra
- Parallel Languages
  - SET Language, *Lisp, and NESL

Limitations and Future Work

- Limitations
  - Mobile Networks
  - Mobile Software Agents
- Future Work
  - Quality of service
  - Automatic adaption to changes

Conclusions

Conclusions

- A user writes simple, easy to understand macroprograms

Conclusions

- A user writes simple, easy to understand macroprograms
- Implementations are chosen automatically for the best performance

MacroLab is available at:

http://www.cs.virginia.edu/hnat/MacroLab/

hnat@cs.virginia.edu

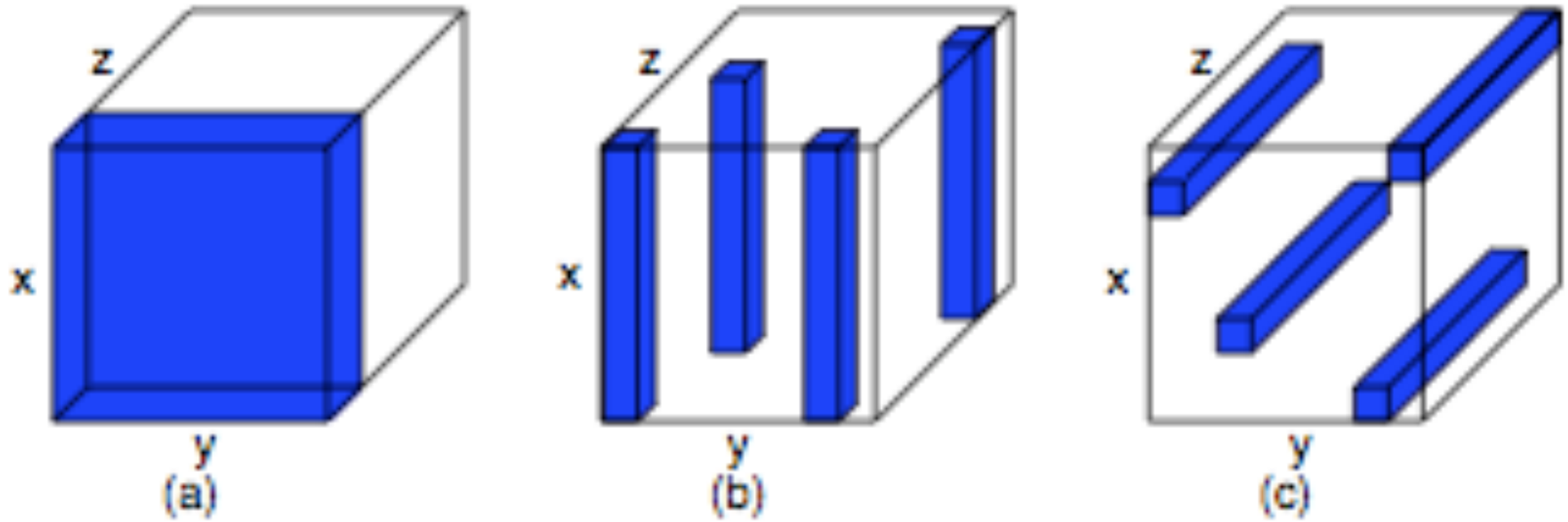
Backup Slides

Actual Breakdown

Actual Breakdown

RTS (ROM/RAM)	MacroLab (ROM/RAM)
558/66	1,264/125
558/66	1,144/24

Dot-Product



Simulation Results

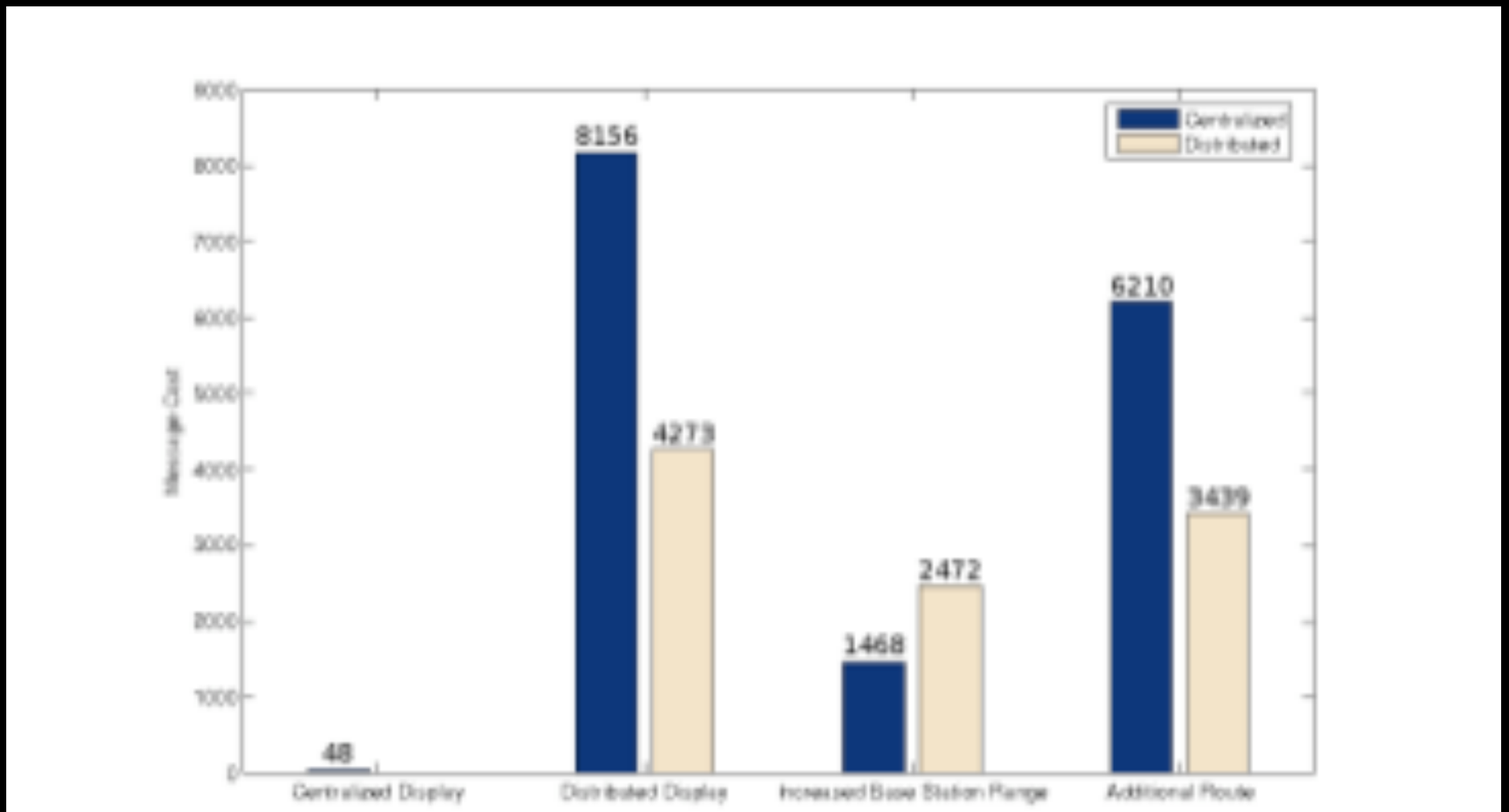


Figure 11. Neither decomposition is best for all deployment scenarios. Small changes in the deployment scenario changes the optimal implementation between centralized and distributed.

Power Measurement

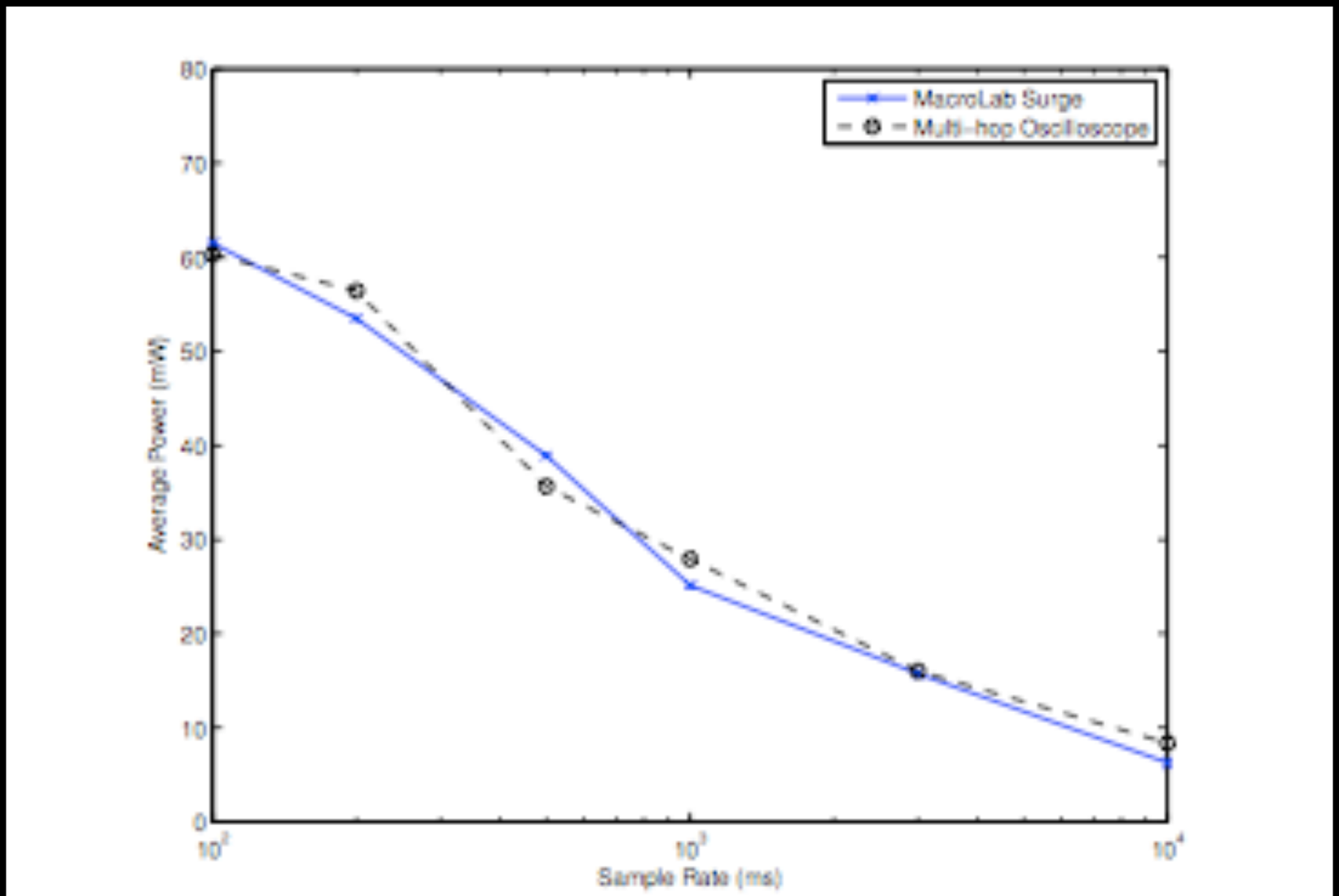


Figure 8. Oscilloscope power measurements of MacroLab and nesC Surge implementations.

Range

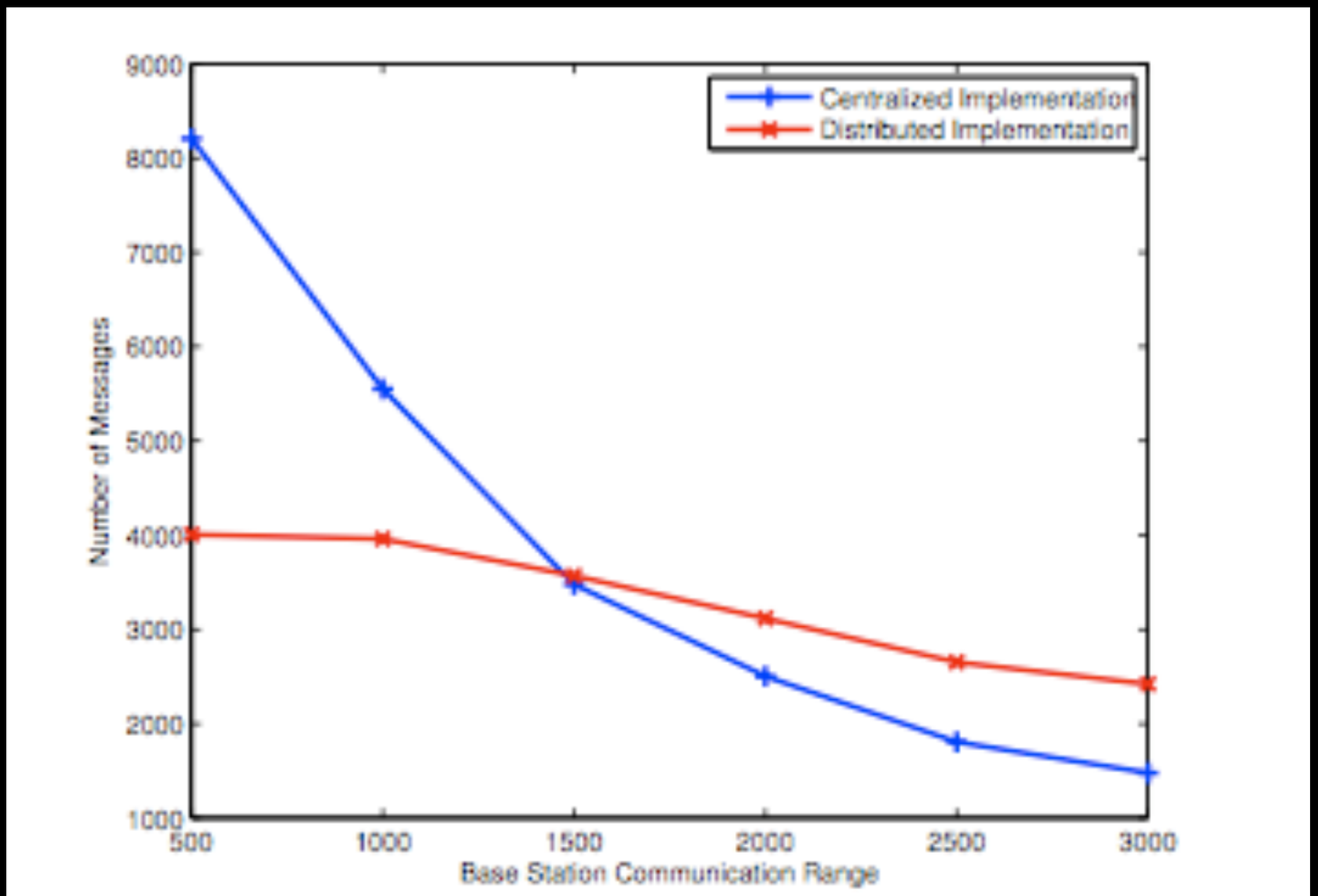


Figure 12. Changing the base station range changes the balance between the two decompositions.

Route

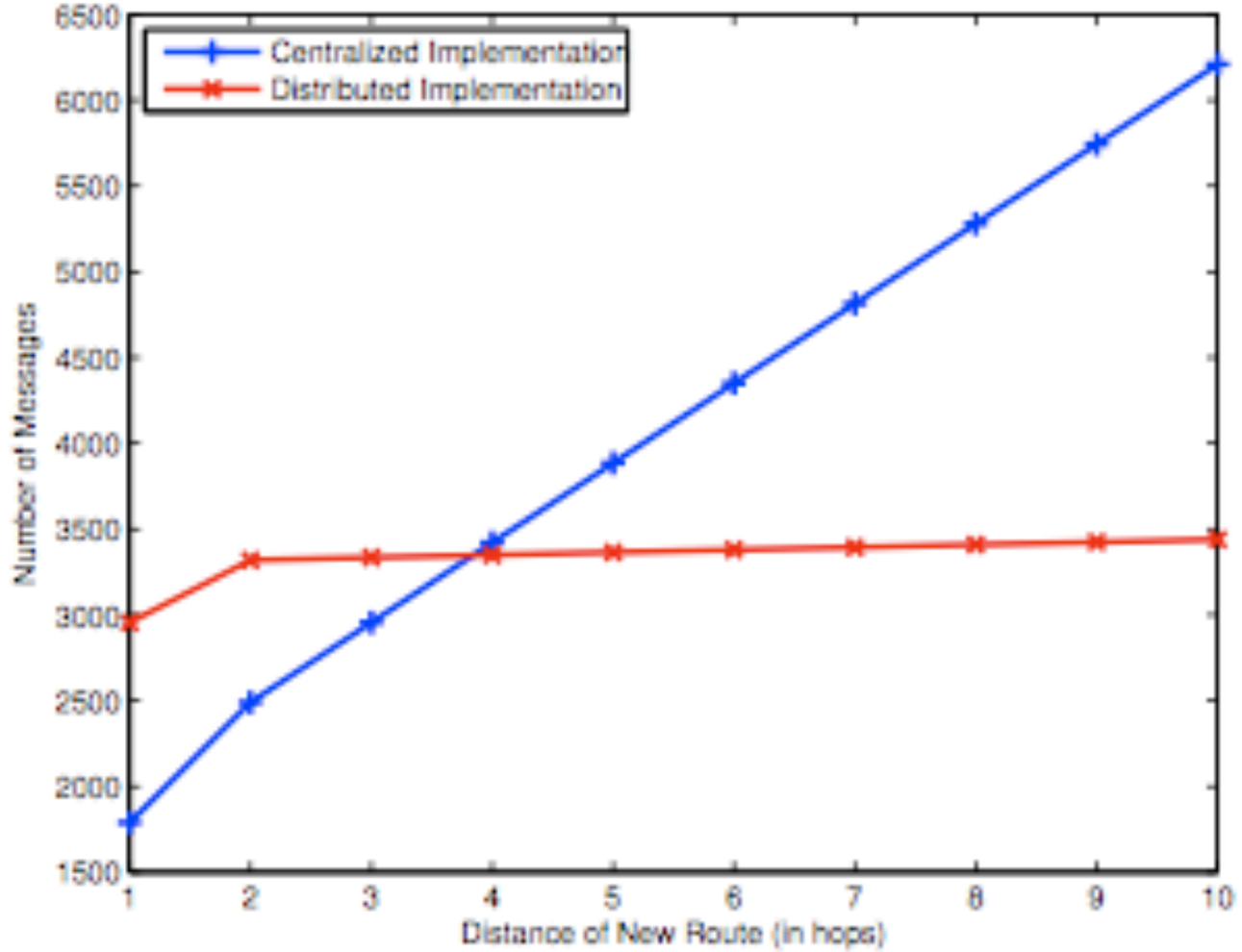


Figure 13. Adding a new route at various distances changes the balance between the two decompositions.

Static Cost

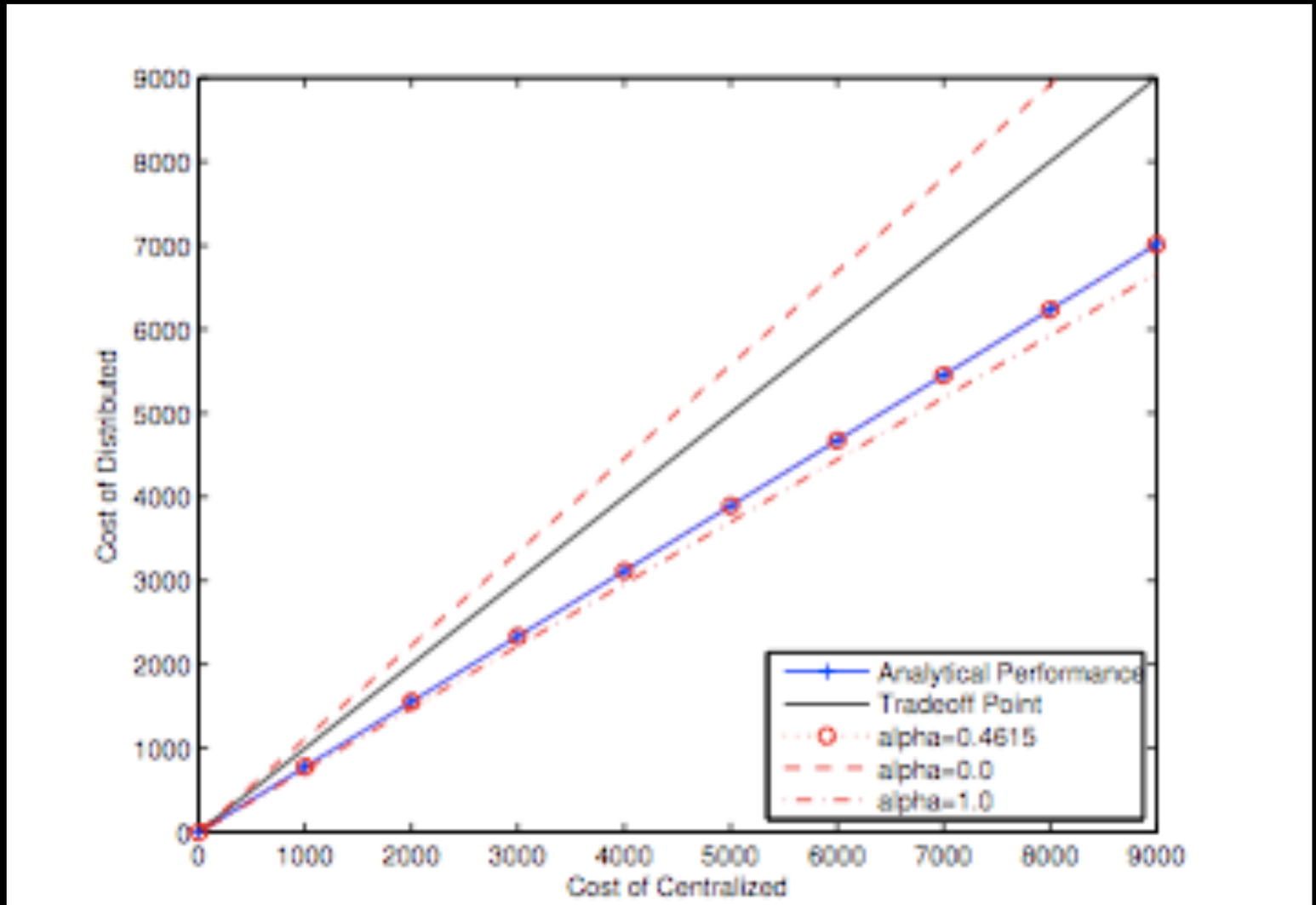


Figure 14. Estimated and measured messaging costs. The parameter α is the ratio of buses on long routes versus those on all other routes.