Korz: Simple, Symmetric, Subjective, Context-Oriented Programming

Harold Ossher, David Ungar and Doug Kimelman

IBM Research

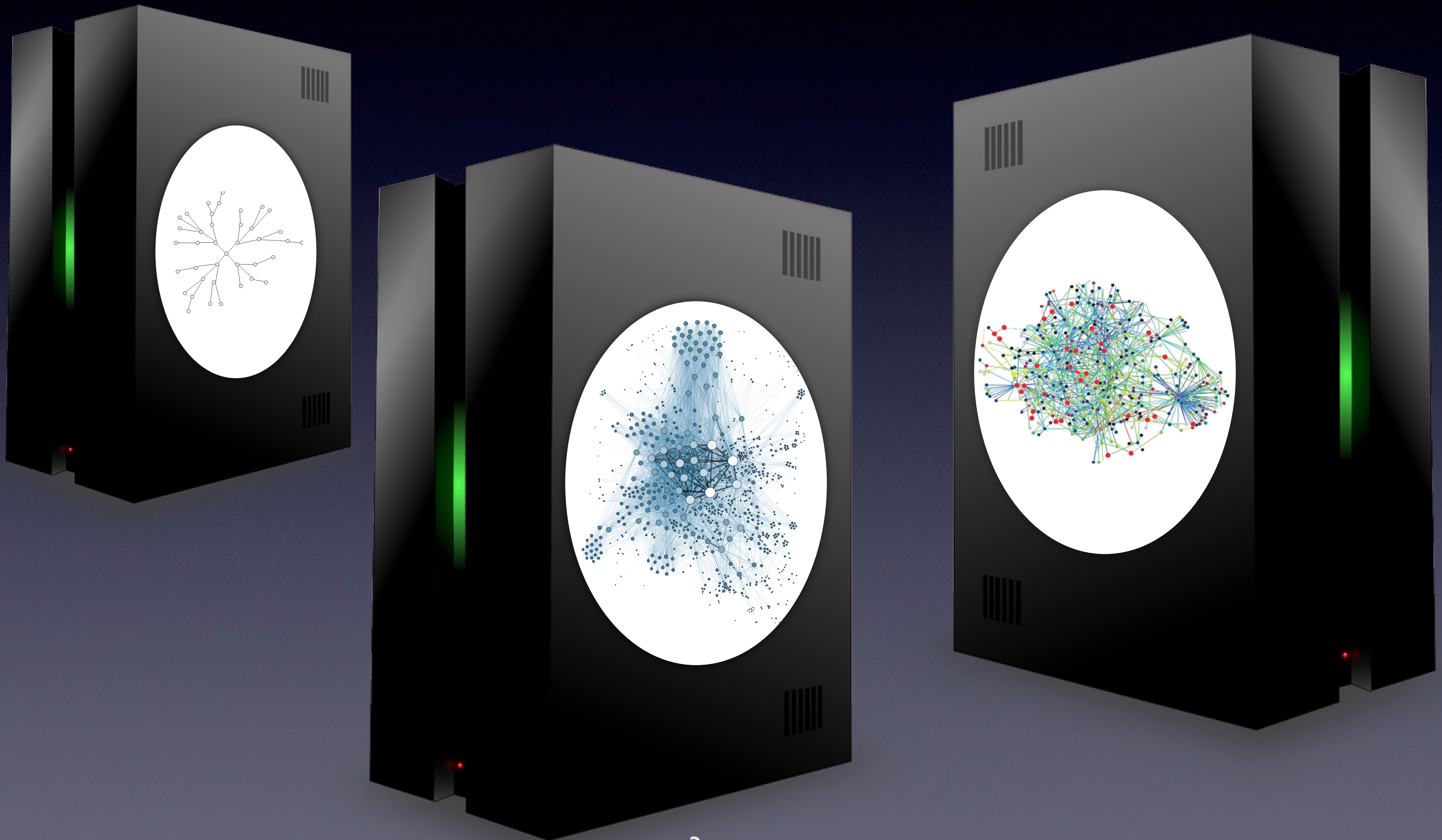
Copyright © IBM Corporation, 2014

Contents

- Motivation: *Dynamic Contextual Variation*
- The Korz model illustrated
- Subjectivity
- Reflection

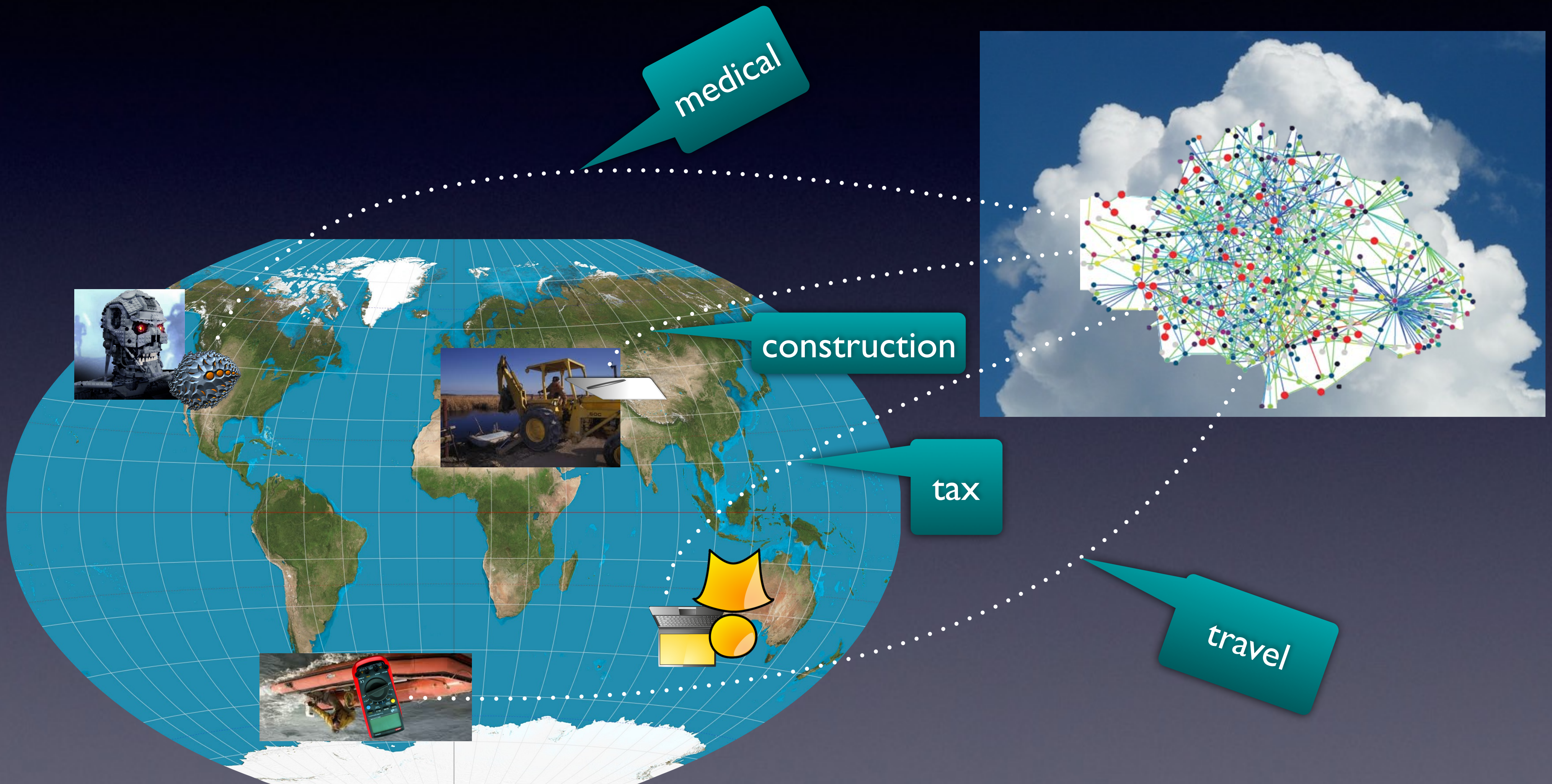
Copyright © IBM Corporation, 2014

Freestanding OO Programs



Copyright © IBM Corporation, 2014

Services, Users, Contexts



Copyright © IBM Corporation, 2014

Dynamic Contextual Variation

- Objects look different in different contexts
  - Both state and behavior
- Context is multi-dimensional
- New context dimensions and dependencies constantly emerge
  - Rapid evolution
  - Dynamic as *cognitive systems* discover relevant context
- Old issues of supporting variation come the fore, with a new twist

Copyright © IBM Corporation, 2014

Displaying a point

point1.display(stdscreen)

pointParent

display(device)	device.drawPixel(x, y, color)

point1

x	0
y	0
color	black

Copyright © IBM Corporation, 2014

Displaying a point

point1.display(stdscreen)

pointParent

display(device)	device.drawPixel(x, y, color)

point1

x	0
y	0
color	black

screenParent

drawPixel(x, y, color)	/* light up a pixel */

stdscreen

...	...

Copyright © IBM Corporation, 2014

Displaying a line

aLine.display(stdscreen)

lineParent

display(device)	for-points-from-start-to-end(p.display(device))

aLine

start	point1
end	point2

pointParent

display(device)	device.drawPixel(x, y, color)

screenParent

drawPixel(x, y, color)	/* light up a pixel */

Copyright © IBM Corporation, 2014

Add Location Sensitivity

aLine.display(stdscreen, S)

lineParent

display(device, hemi)	for-points-from-start-to-end(p.display(device, hemi))

aLine

start	point1
end	point2

pointParent

display(device, hemi)	device.drawPixel(x, y, color, hemi)

screenParent

drawPixel(x, y, color, hemi)	drawPixel(x, hemi = S ? -y : y, color)
drawPixel(x, y, color)	/* light up a pixel */

Copyright © IBM Corporation, 2014

Implicit context flow: *self* object

aLine.display(stdscreen, S)

self: aLine

lineParent

display(device, hemi)	for-points-from-start-to-end(p.display(device, hemi))

self: p

aLine

start	point1
end	point2

pointParent

display(device, hemi)	device.drawPixel(x, y, color, hemi)

self: stdscreen

screenParent

drawPixel(x, y, color, hemi)	drawPixel(x, hemi = S ? -y : y, color)
drawPixel(x, y, color)	/* light up a pixel */

self: stdscreen

Copyright © IBM Corporation, 2014

Context Clutter

aLine.display(stdscreen, S)

lineParent

display(device, **hemi**)	for-points-from-start-to-end(p.display(device, **hemi**))

aLine

start	point1
end	point2

pointParent

display(device, **hemi**)	device.drawPixel(x, y, color, **hemi**)

screenParent

drawPixel(x, y, color, **hemi**)	drawPixel(x, hemi = S ? -y : y, color)
drawPixel(x, y, color)	/* light up a pixel */

Context passed *explicitly* through methods that don't care

Copyright © IBM Corporation, 2014

Context Clutter

aLine.display(stdscreen, S)

lineParent

display(**device**, **hemi**) | for-points-from-start-to-end(p.display(**device**, **hemi**))

aLine

start	point1
end	point2

pointParent

display(**device**, **hemi**) | device.drawPixel(x, y, color, **hemi**)

screenParent

drawPixel(x, y, color, **hemi**) | drawPixel(x, hemi = S ? -y : y, color)

drawPixel(x, y, color) | /* light up a pixel */

Context passed *explicitly* through methods that don't care

Copyright © IBM Corporation, 2014

Variation Explosion

aLine.display(stdscreen, **S, …**)

lineParent

display(device, **hemi, …**)	for-points-from-start-to-end(p.display(device, **hemi, …**))

...

aLine

start	point1
end	point2

pointParent

display(device, hemi)	device.drawPixel(x, y, color, **hemi, …**)

...

screenParent

drawPixel(x, y, color, hemi, …)	drawPixel(x, hemi = S ? -y : y, color, **…**)
drawPixel(x, y, color)	/* light up a pixel */

...

Every dimension of variation creates another problem

Copyright © IBM Corporation, 2014

Dynamic Contextual Variation

requires a

new way of thinking

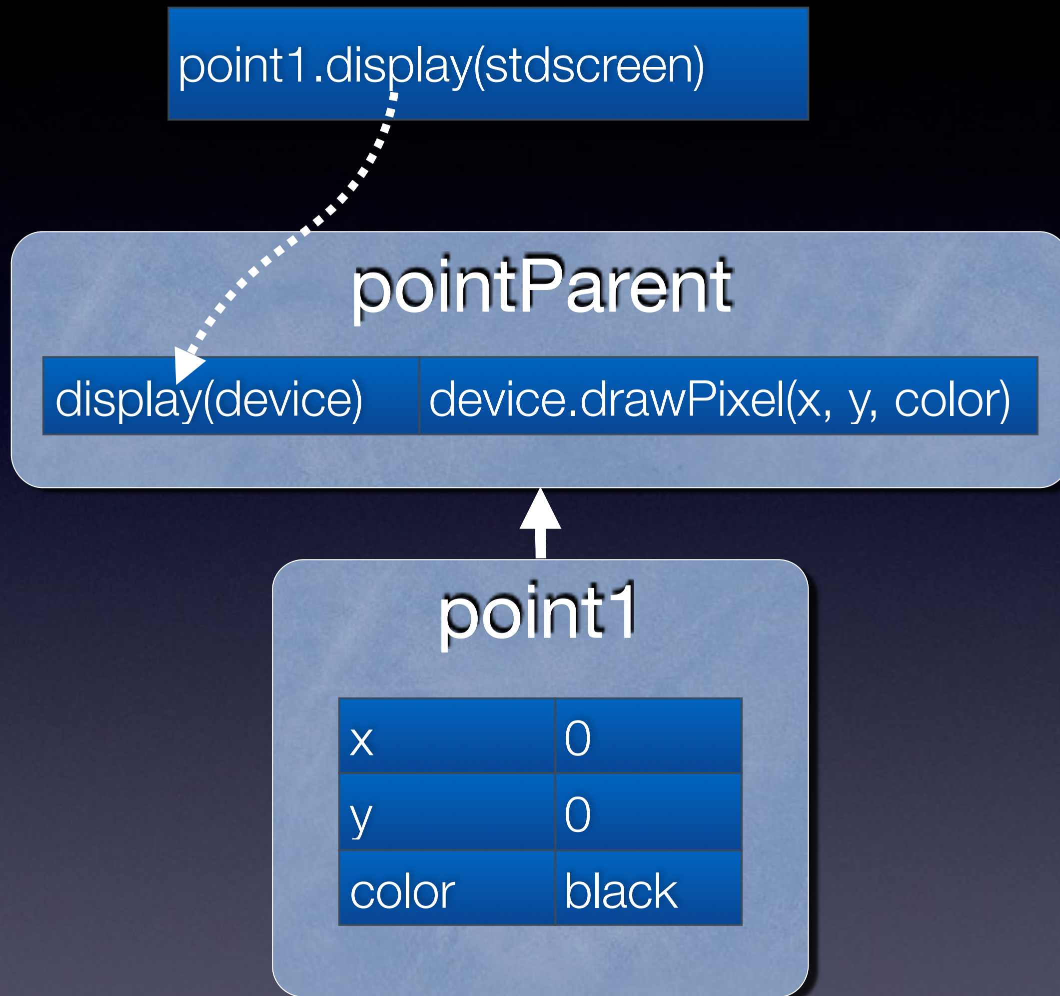
about objects

Copyright © IBM Corporation, 2014

Dynamic Contextual Variation

Copyright © IBM Corporation, 2014

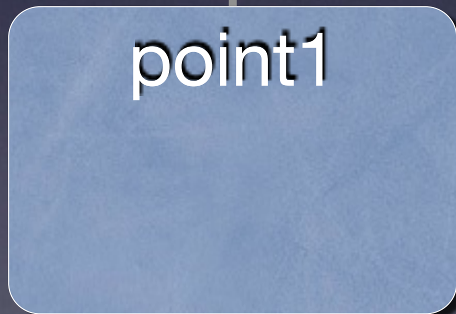
Old: Slots in objects



Copyright © IBM Corporation, 2014

New: Slots linked instead of contained

point1.display(stdscreen)



Copyright © IBM Corporation, 2014

Objects are just unique coordinates, with identity

point1.display(stdscreen)

pointParent

rcvr	display(device	device.drawPixel(x, y, color)

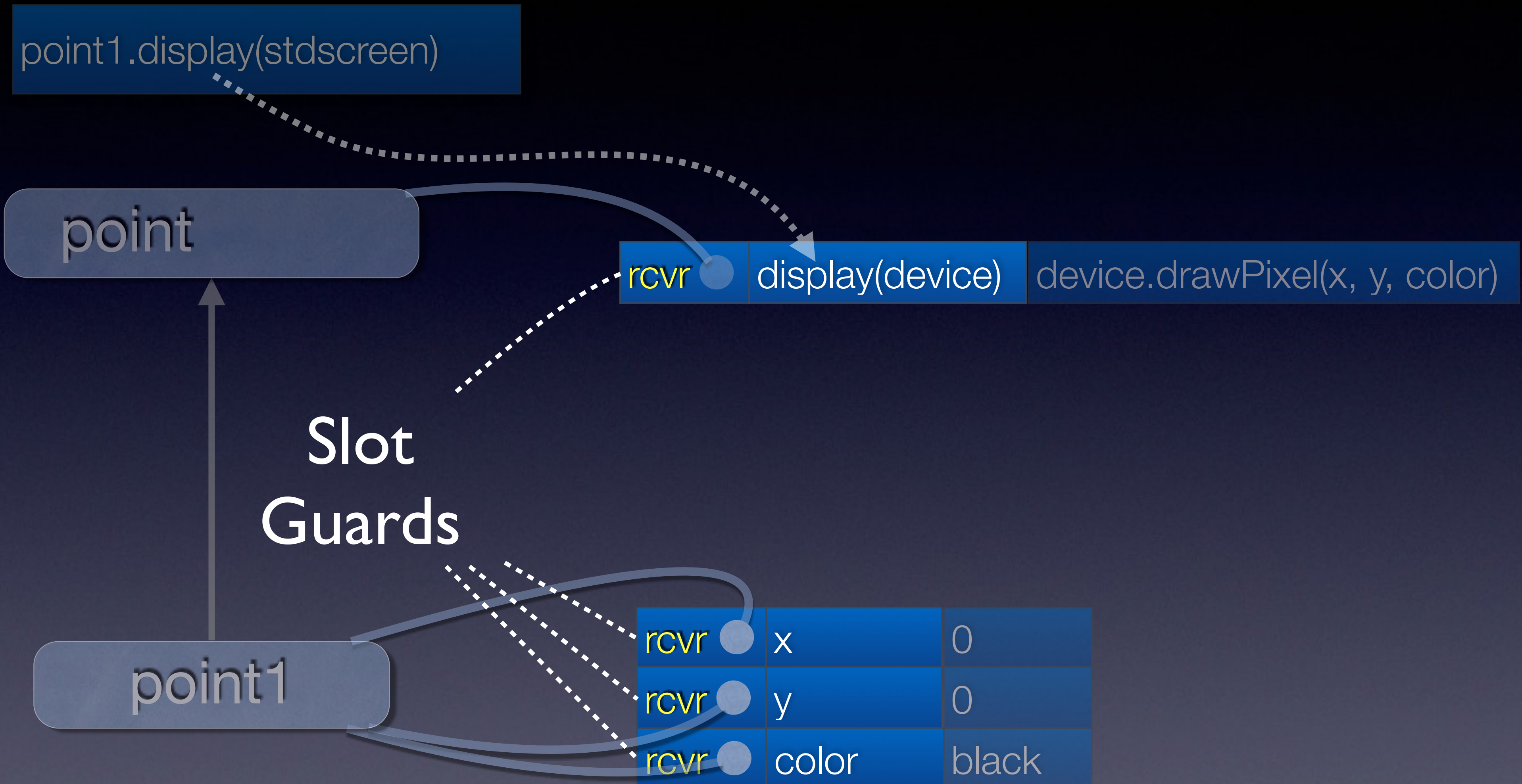
point1

rcvr	x	0
rcvr	y	0
rcvr	color	black

Coordinates

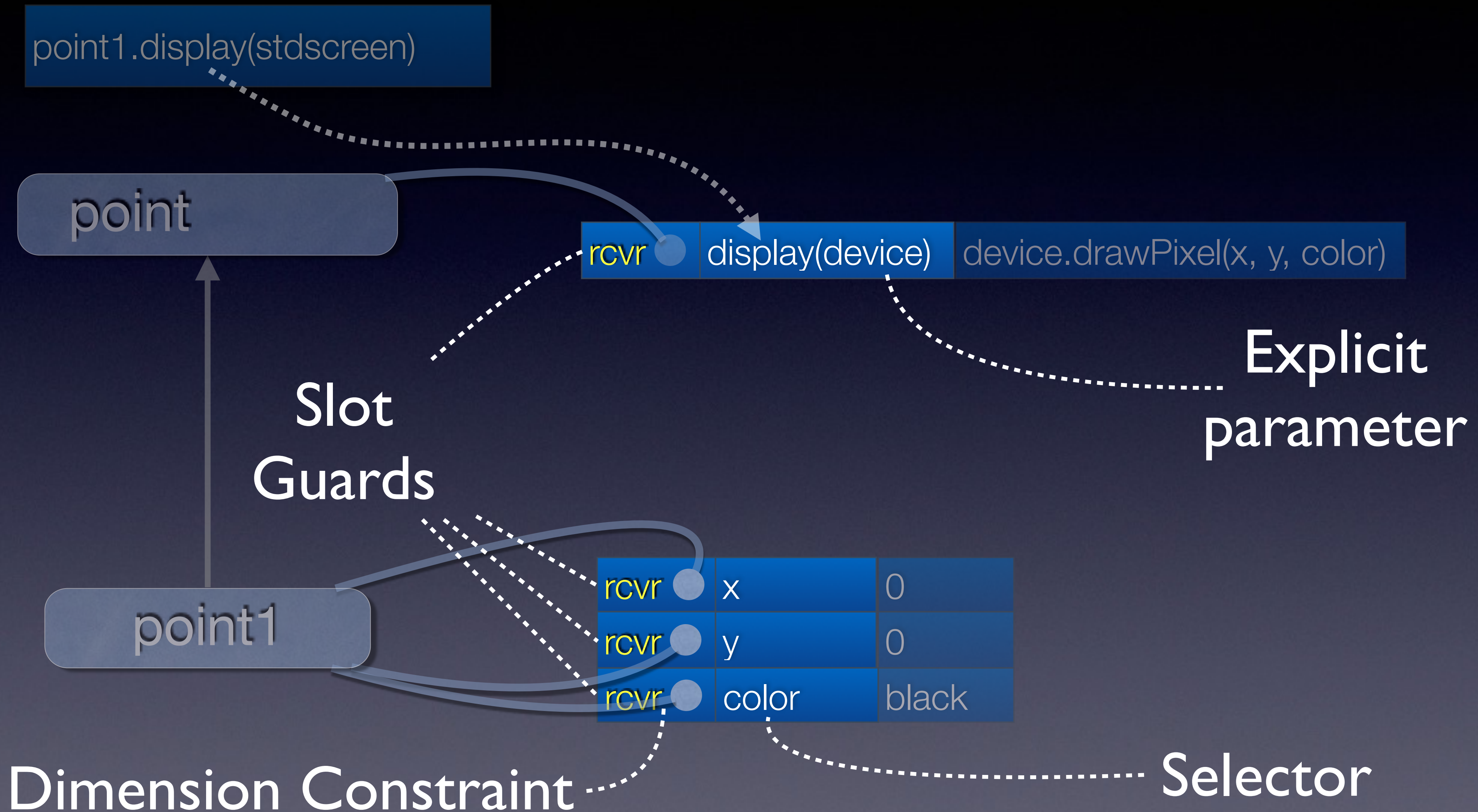
Copyright © IBM Corporation, 2014

Slot guards control dispatch



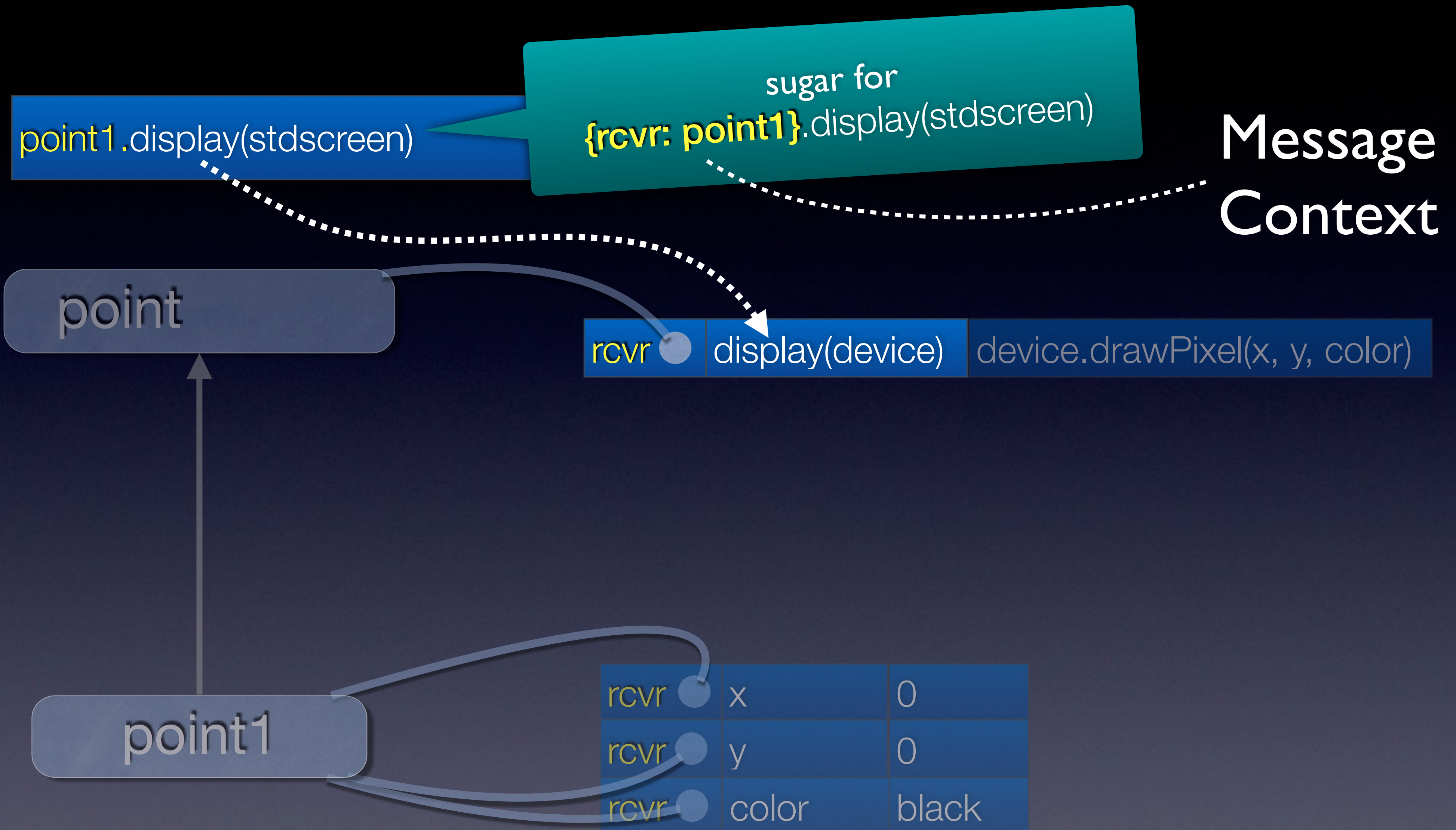
Copyright © IBM Corporation, 2014

Slot guards control dispatch



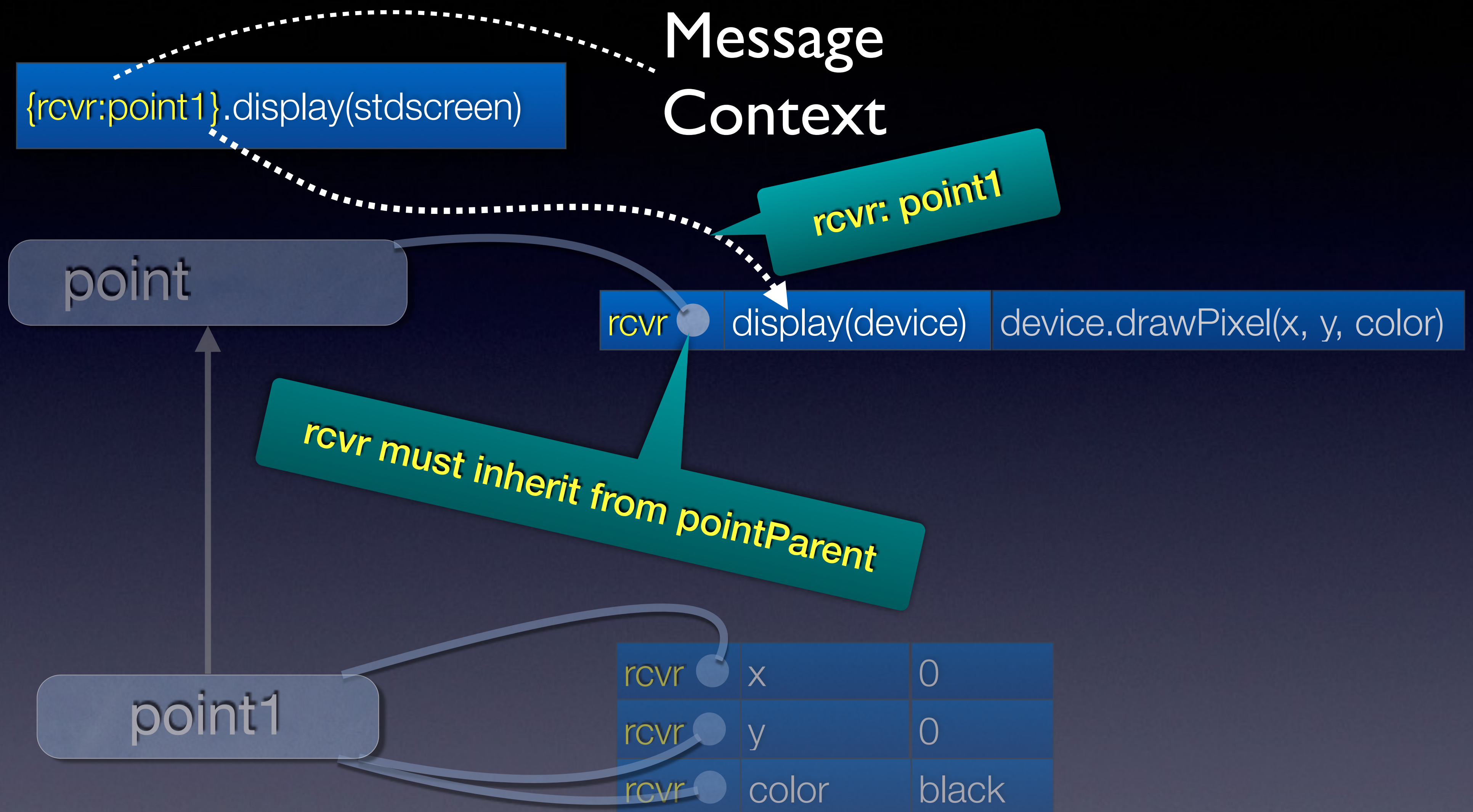
Copyright © IBM Corporation, 2014

Message Context



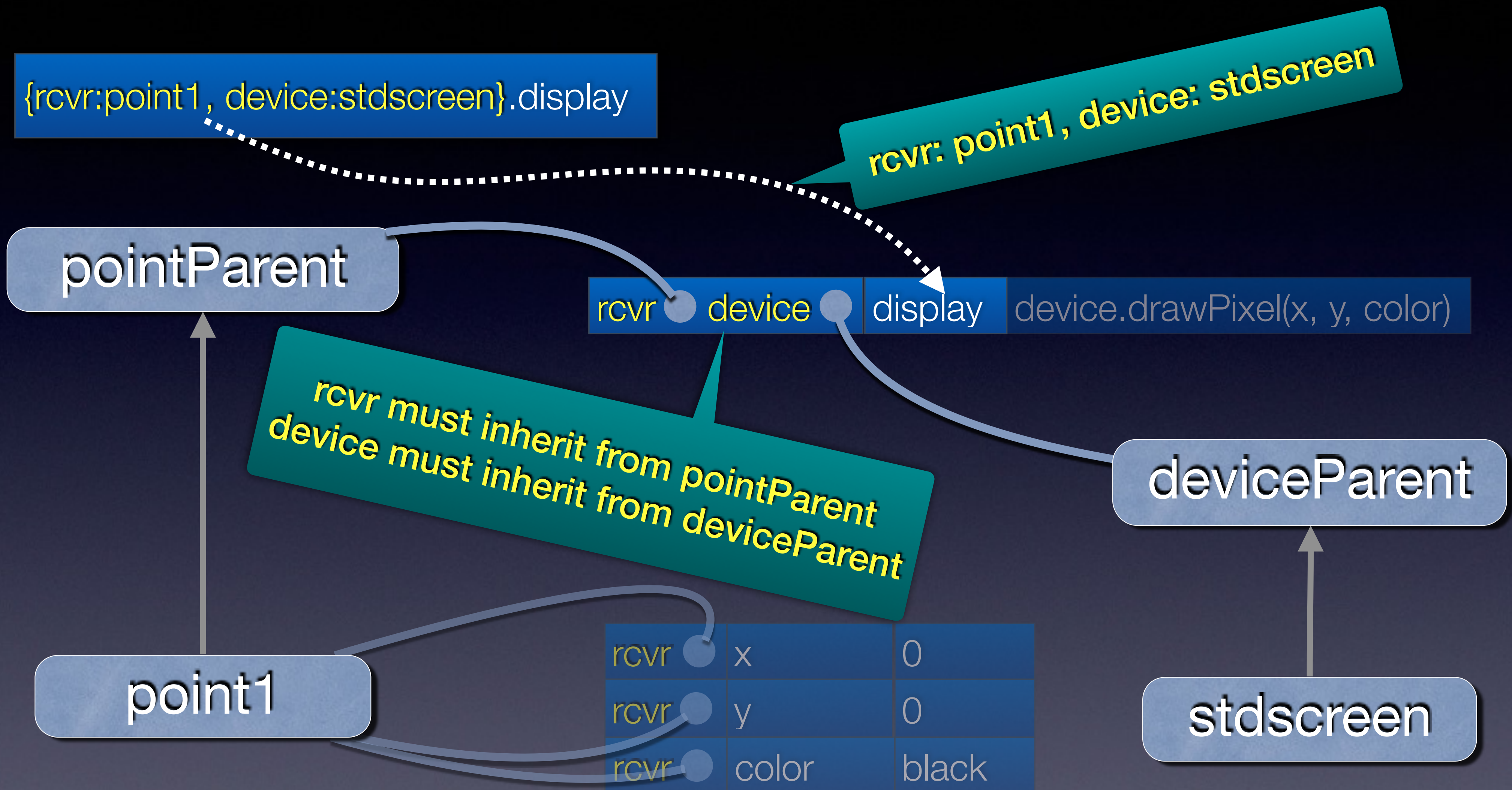
Copyright © IBM Corporation, 2014

Slot Guard must match Context



Copyright © IBM Corporation, 2014

More Implicit Context, and Multiple Dispatch



Coordinates in all dimensions must match

Copyright © IBM Corporation, 2014

Simplify picture: Show only the parents

{rcvr:point1, device:stdscreen}.display

pointParent

rcvr	device	display	device.drawPixel(x, y, color)

deviceParent

Copyright © IBM Corporation, 2014

Simplify picture: Show only the parents

{rcvr:point1, device:stdscreen}.display

pointParent

rcvr	device	display	device.drawPixel(x, y, color)

deviceParent

Copyright © IBM Corporation, 2014

Full Example

{rcvr:aLine, device:stdscreen}.display

lineParent

rcvr	display	for-points-from-start-to-end(p.display())

pointParent

rcvr	device	display	device.drawPixel(x, y, color)

deviceParent

rcvr	drawPixel(x, y, color)	/* light up a pixel */

Copyright © IBM Corporation, 2014

Add Location Sensitivity

{rcvr:aLine, device:stdscreen}

lineParent

rcvr	display	for-points-from-start-to-end(p.display())

point

rcvr	device	display	device.drawPixel(x, y, color)

S

rcvr	hemi	drawPixel(x, y, color)	{-hemi}.drawPixel(x, -y, color)

device

rcvr	drawPixel(x, y, color)	/* light up a pixel */

Just add one coordinate and one slot

Copyright © IBM Corporation, 2014

Add Location Sensitivity

{rcvr:aLine, device:std, hemi:S}.display

lineParent

rcvr	display	for-points-from-start-to-end(p.display())

point

rcvr	device	display	device.drawPixel(x, y, color)

S

rcvr	hemi	drawPixel(x, y, color)	{-hemi}.drawPixel(x, -y, color)

device

rcvr	drawPixel(x, y, color)	/* light up a pixel */

Just add one coordinate and one slot

Copyright © IBM Corporation, 2014

Dimensions not mentioned in slot guard match

{rcvr:aLine, device:std, hemi:S}.display

rcvr: aLine, device: std, hemi: S

lineParent

rcvr	display	for-points-from-start-to-end(

point

rcvr	device	display	device.drawPixel(x, y, color)

S

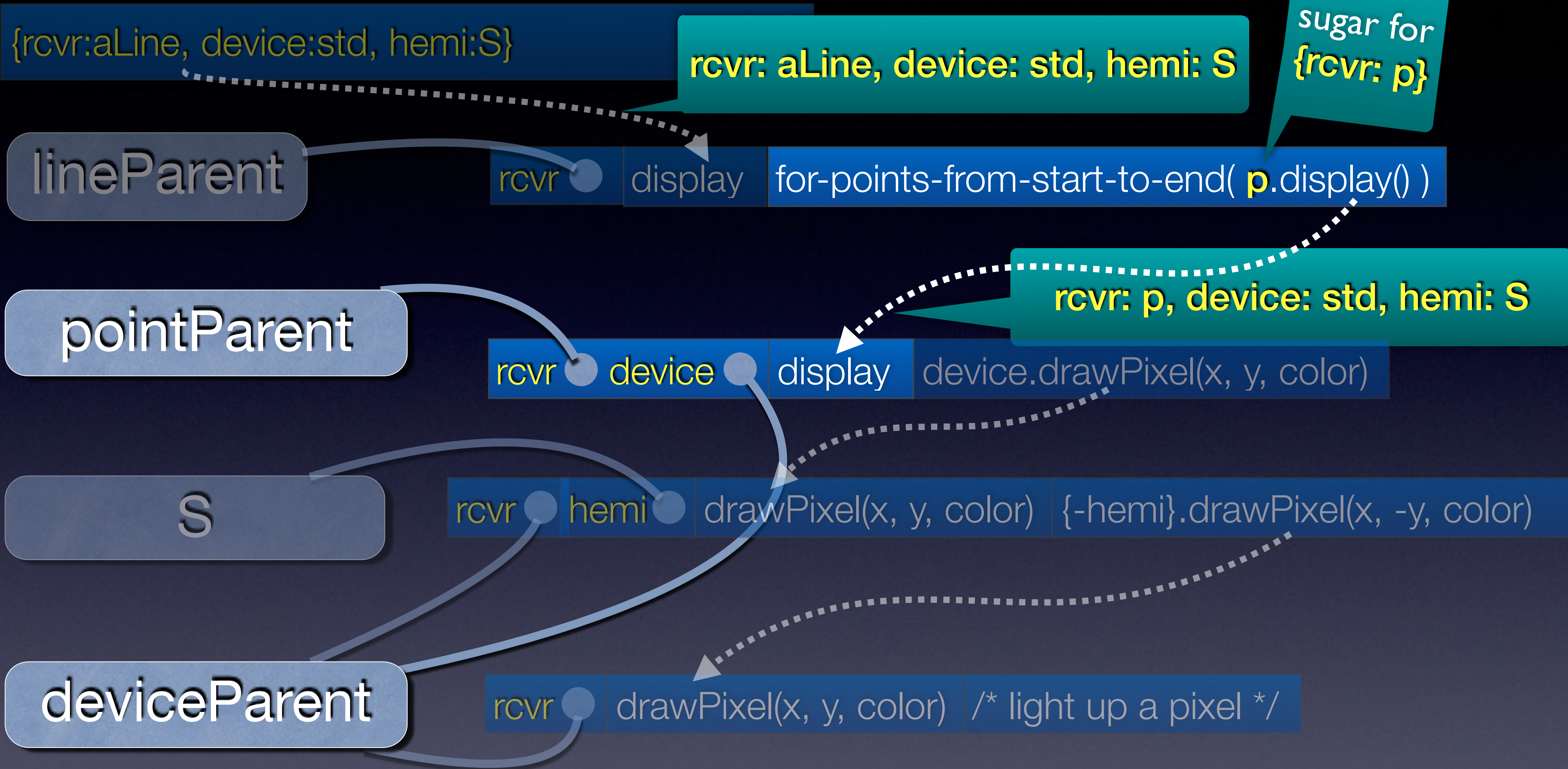
rcvr	hemi	drawPixel(x, y, color)	{-hemi}.drawPixel(x, -y, color)

device

rcvr	drawPixel(x, y, color)	/* light up a pixel */

Copyright © IBM Corporation, 2014

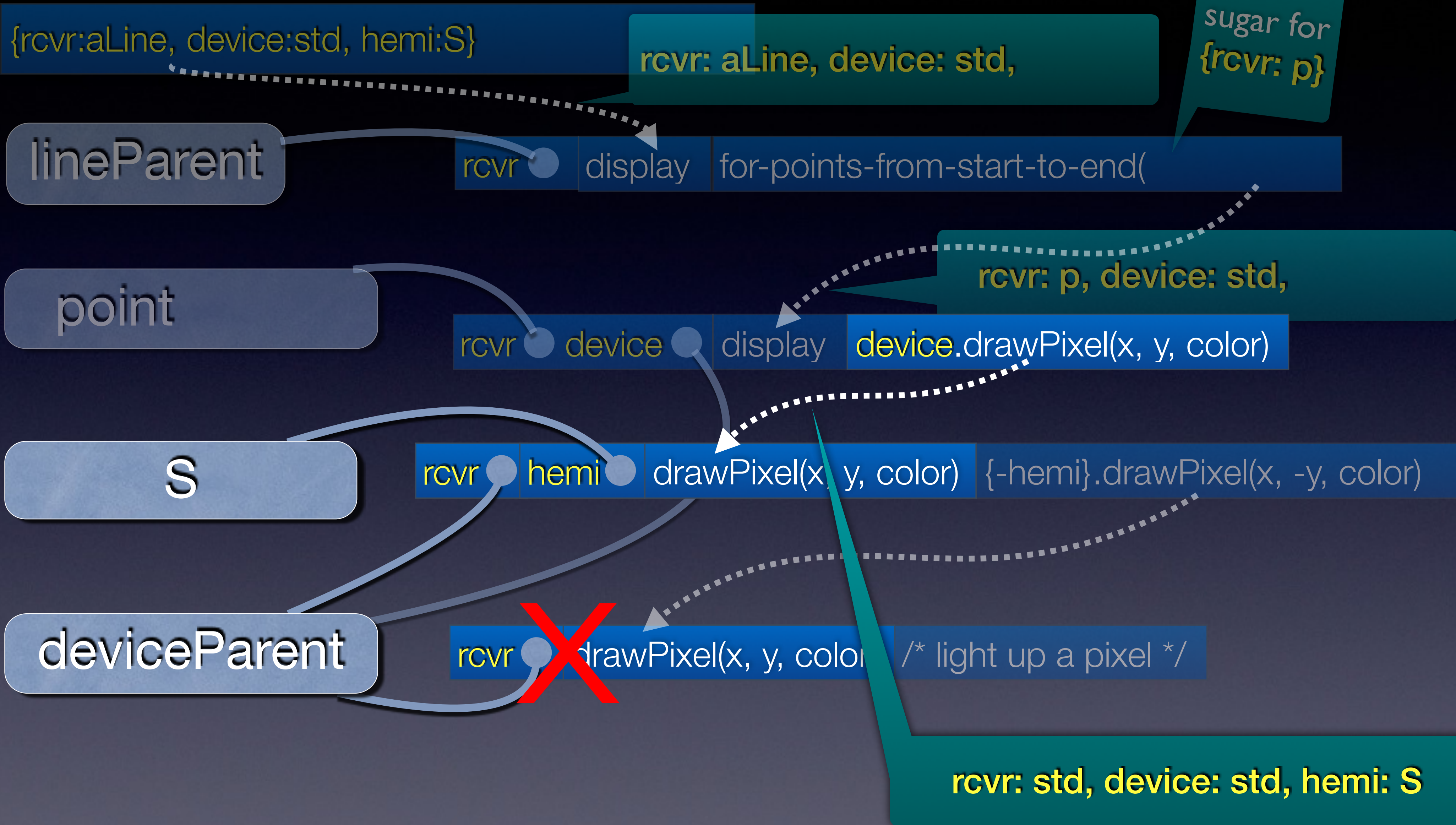
Implicit parameters propagated automatically



Can be explicitly changed

Copyright © IBM Corporation, 2014

Most-specific match wins



Copyright © IBM Corporation, 2014

All slot-guard dimensions must match

{rcvr:aLine, device:std, hemi:S}

rcvr: aLine, device: std,

sugar for {rcvr: p}

lineParent

rcvr | display | for-points-from-start-to-end(

rcvr: p, device: std,

point

rcvr | device | display | device.drawPixel(x, y, color)

S

rcvr | hemi | drawPixel(x, y, color) | {-hemi}.drawPixel(x, -y, color)

deviceParent

rcvr | drawPixel(x, y, color) | /* light up a pixel */

rcvr: std, device: std,

rcvr: std, device: std

Copyright © IBM Corporation, 2014

Adding hemisphere was easy

- Did not need:
  - global
  - extra parameter everywhere
  - visitor or strategy
  - pluggable objects, aspects, …

But, what about a second, different kind of dimension?

Copyright © IBM Corporation, 2014

Add another dimension: *tint*

{rcvr:aLine, device:std, tint:[0.7, 1.0, 1.0]}.display

Any 3-element vector of numbers, can be created dynamically

lineParent

rcvr | display | for-points-from-start-to-end(p.display())

point

rcvr | device | display | device.drawPixel(x, y, color)

S

rcvr | hemi | drawPixel(x, y, color) | {-hemi}.drawPixel(x, -y, color)

device

rcvr | drawPixel(x, y, color) | /* light up a pixel */

Copyright © IBM Corporation, 2014

Add another dimension: *tint*

{rcvr:aLine, device:std, tint:[0.7, 1.0, 1.0]}.display

lineParent

rcvr | display | for-points-from-start-to-end(p.display())

point

rcvr | device | display | device.drawPixel(x, y, color)

S

rcvr | hemi | drawPixel(x, y, color) | {-hemi}.drawPixel(x, -y, color)

vectorParent

rcvr | tint | drawPixel(x, y, color)

{-tint}.drawPixel(x, y, color.adjust(tint))

deviceParent

rcvr | drawPixel(x, y, color) | /* light up a pixel */

Copyright © IBM Corporation, 2014

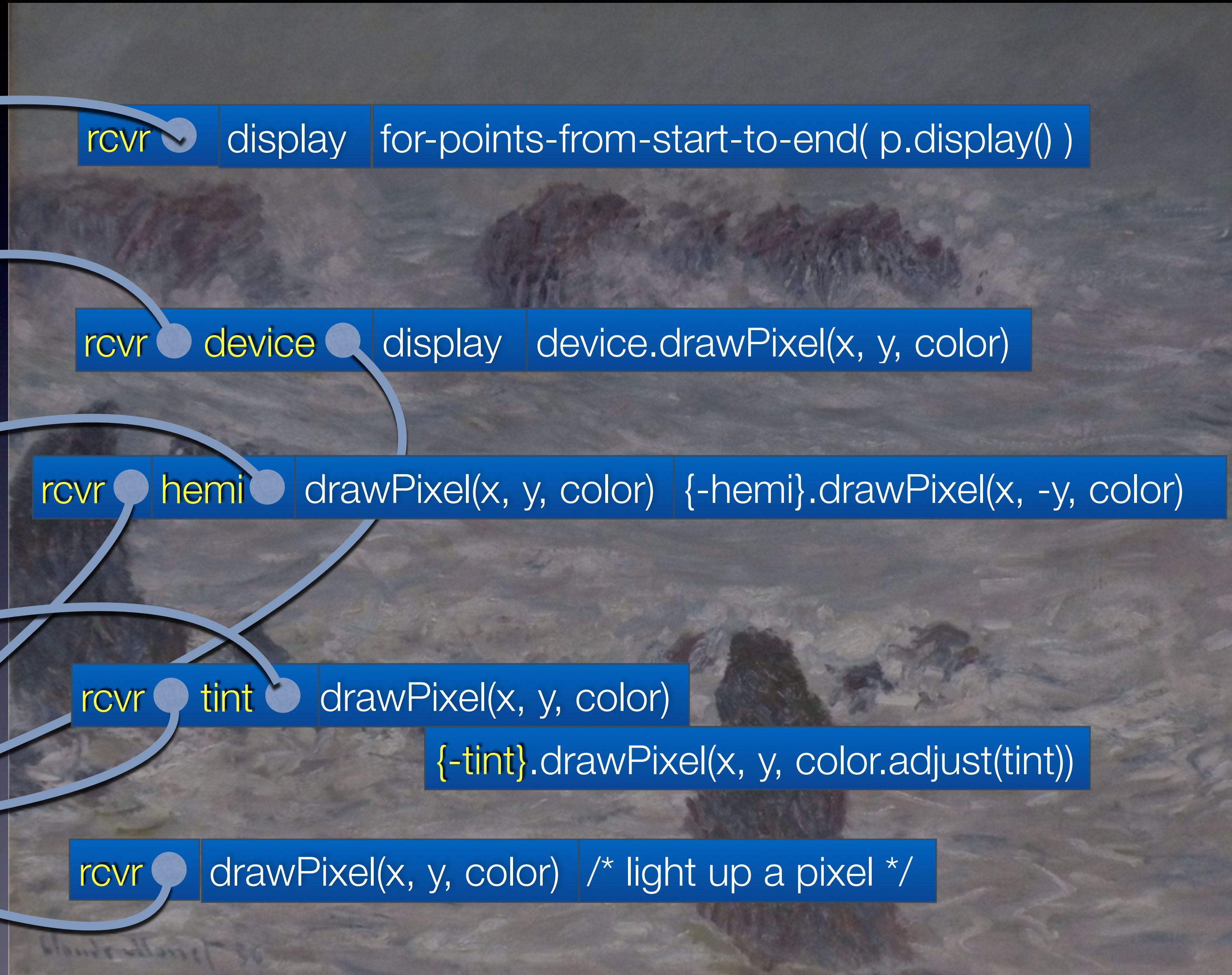
See the Onward! 2014 paper for …

- Language definition
- Prototype implementation
- Analysis of related work
- Discussion

David Ungar, Harold Ossher and Doug Kimelman,
"Korz: Simple, Symmetric, Subjective, Context-Oriented Programming."
In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, ACM, pages 113-131, October 2014.
http://dl.acm.org/citation.cfm?id=2661147

Copyright © IBM Corporation, 2014

Sea of slots



Copyright © IBM Corporation, 2014

Figure “objects”

lineParent

rcvr: ●	display	for-points-from-start-to-end(p.display())

pointParent

rcvr: ● device: std	display	device.drawPixel(x, y, color)

Copyright © IBM Corporation, 2014

Sea of slots

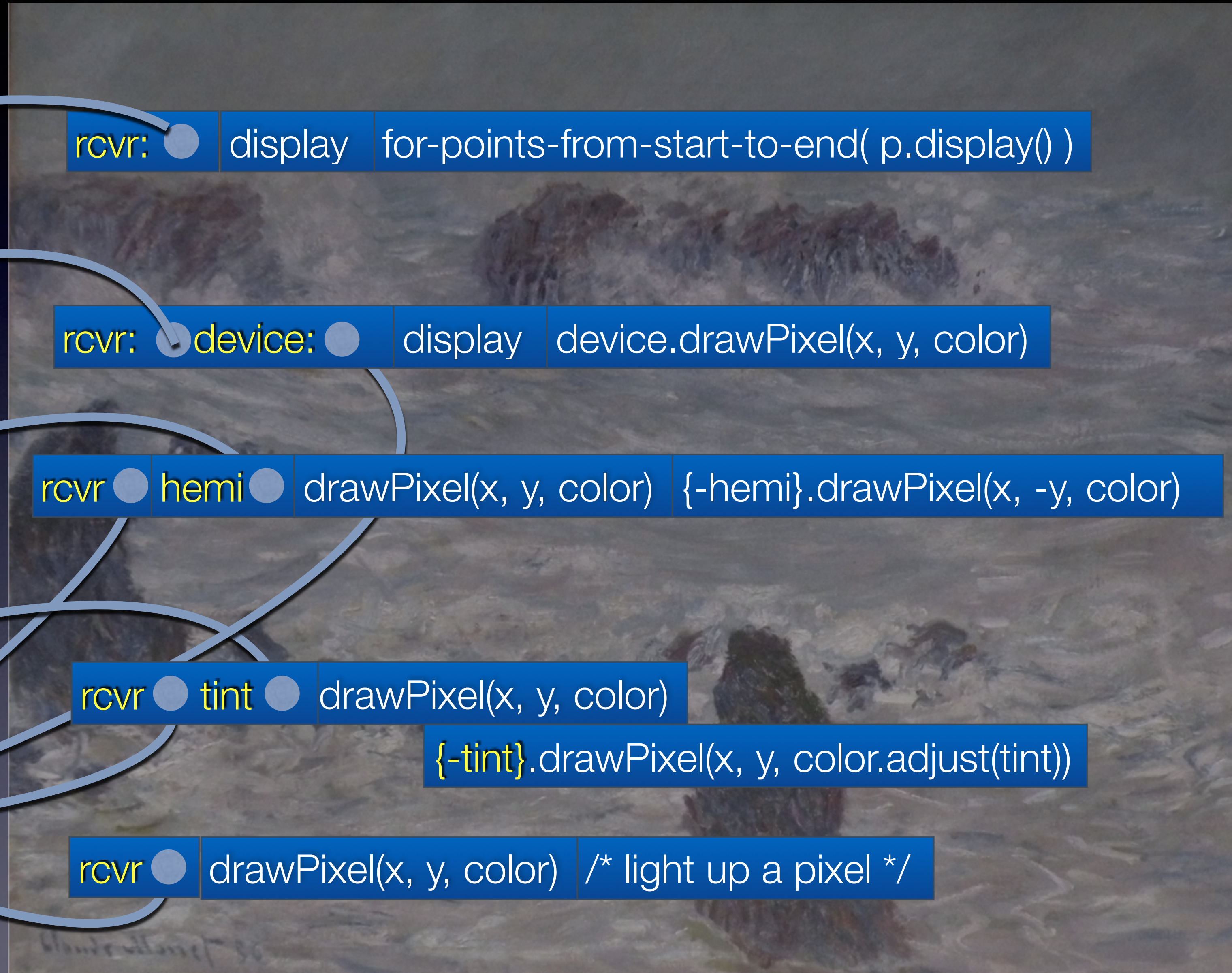
lineParent

pointParent

S

vectorParent

deviceParent



Copyright © IBM Corporation, 2014

Device "objects"

deviceParent

rcvr: pointParent,	device ●	display	device.drawPixel(x, y, color)

rcvr ●	hemi: S	drawPixel(x, y, color)	{-hemi}.drawPixel(x, -y, color)

rcvr ●	tint: vectorParent	drawPixel(x, y, color)

{-tint}.drawPixel(x, y, color.adjust(tint))

rcvr ●	drawPixel(x, y, color)	/* light up a pixel */

Copyright © IBM Corporation, 2014

Sea of slots

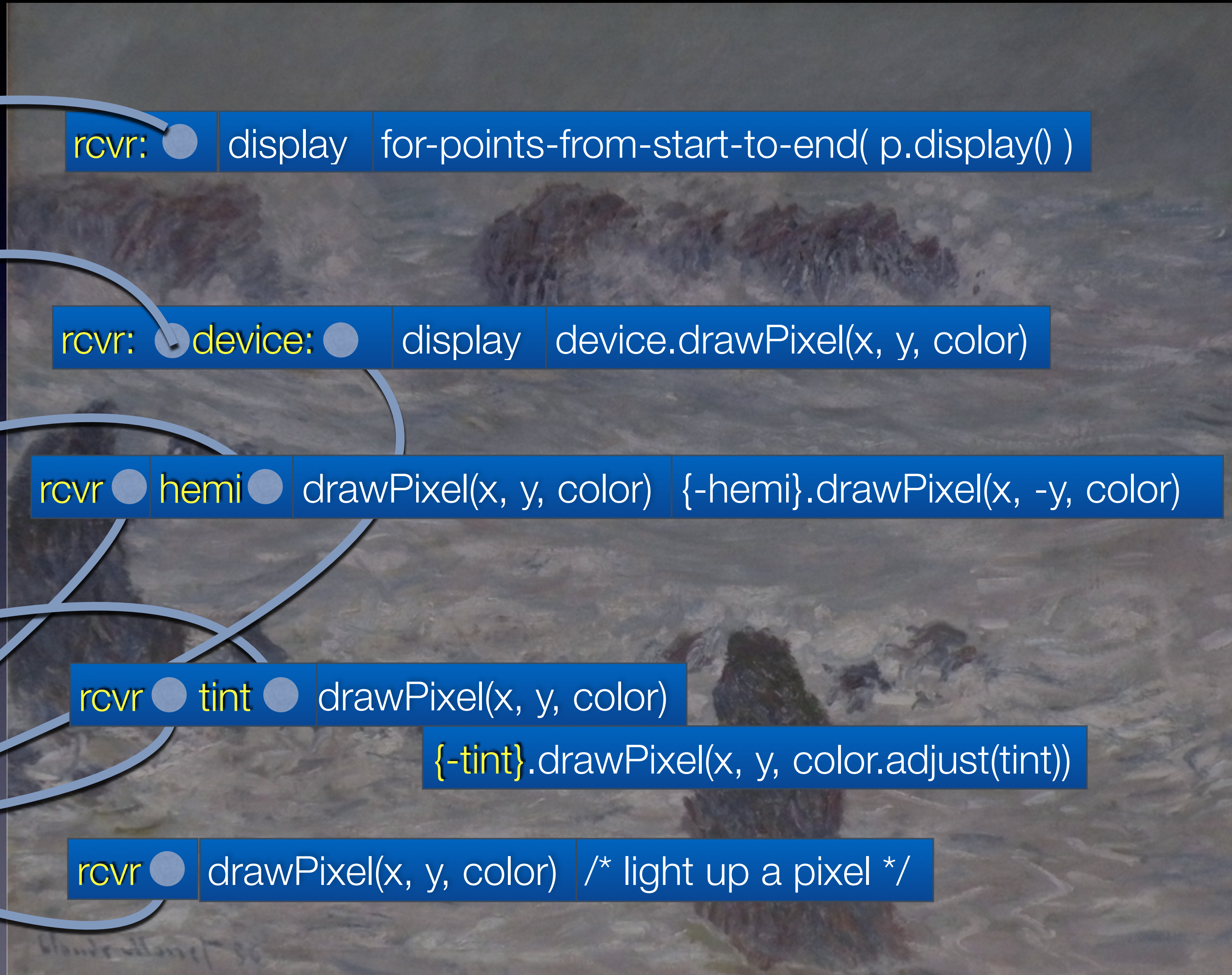
lineParent

pointParent

S

vectorParent

deviceParent



Copyright © IBM Corporation, 2014

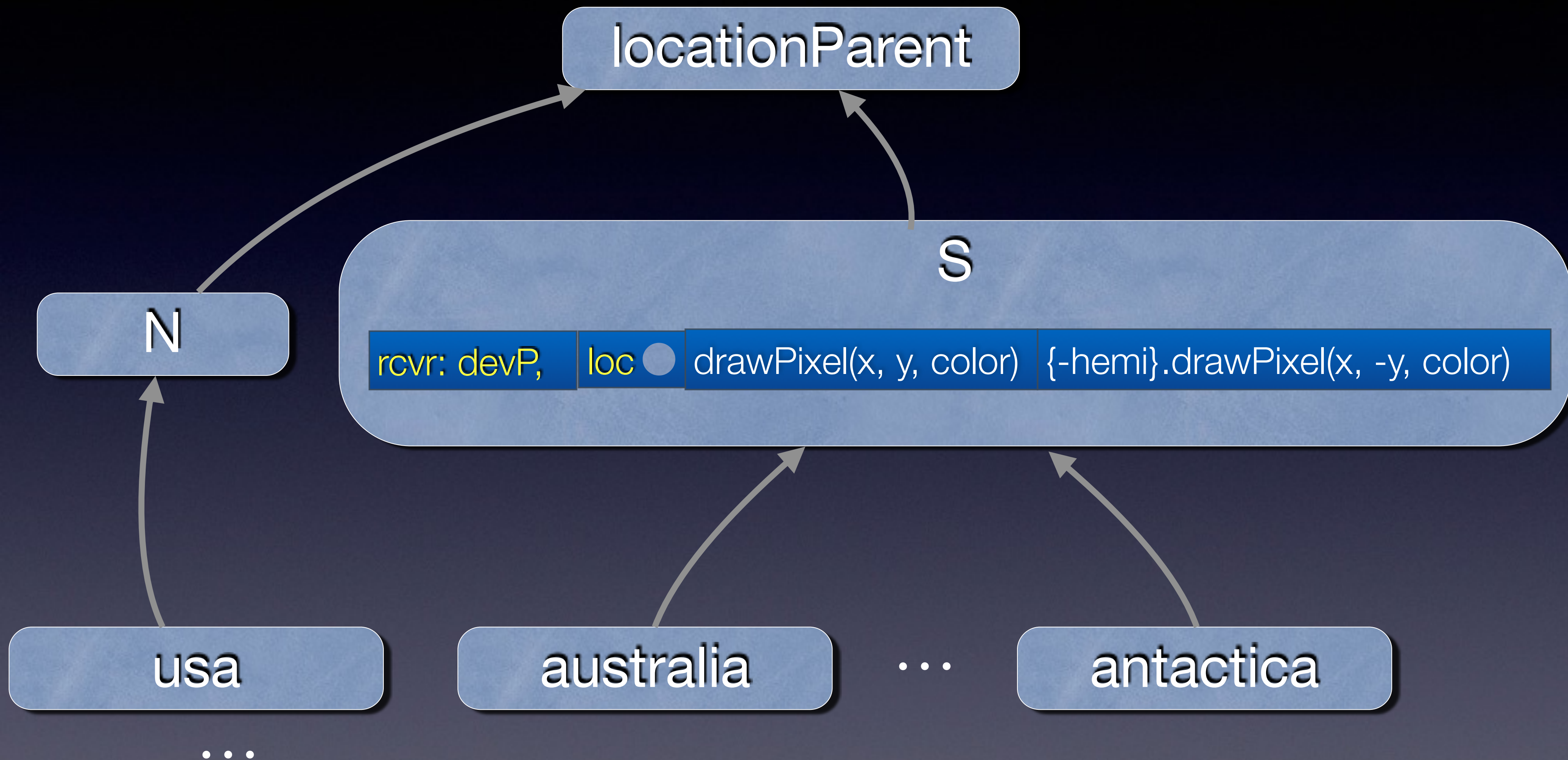
Hemisphere "object"

S

rcvr: devP,	hemi	drawPixel(x, y, color)	{-hemi}.drawPixel(x, -y, color)

Copyright © IBM Corporation, 2014

Generalize to Location Hierarchy



Copyright © IBM Corporation, 2014

Subjectivity

- Sea of slots
- Object-like views gather slots into “objects”
- Different views for different contexts

Copyright © IBM Corporation, 2014

Reflection with Mirrors

Essential problem: Schizophrenia✳

Every time one object hands something to another, should it pass a *mirror* or an *object?*

[Randy Smith, prophetically]

- Often not a problem
- But when it is…

Huge problem in practice!

Copyright © IBM Corporation, 2014

Better reflection: Two views, not two objects

aPoint (base-level view)

parent*	. . .
x	3
y	4
✳ arg	*return* (rho ✳ arg rho) angle: . . .
rho	*return* ((x✳x)+(y✳y)) sqrt
theta	*return* y arctan: x

aPoint (source view)

✳ arg	"(rho ✳ arg rho) angle: . . ."
rho	"((x✳x)+(y✳y)) sqrt"
theta	"y arctan: x"

Different views in different contexts

How?

Copyright © IBM Corporation, 2014

Add 'view' & a viewing slot

sourceView

view	rcvr	_	Return source code of matching slot

Copyright © IBM Corporation, 2014

Multiple links: most matches wins

{rcvr: aPoint, view: sourceView}.rho

sourceView

view	rcvr	_	Return source code of matching slot

Most matches when view = sourceView:
rcvr matches anything, selector matches anything, view matches sourceView

aPoint

rcvr	x	3
rcvr	y	4
rcvr	✳ arg	return (rho ✳ arg rho) angle: (theta + arg theta)
rcvr	rho	return ((x✳x)+(y✳y)) sqrt
rcvr	theta	return y arctan: x

Same reference for aPoint,
base or meta.

Copyright © IBM Corporation, 2014

Selector can be a dimension

{rcvr: aPoint, view: sourceView, selector: “rho”}

sourceView

view	rcvr	_	Return source code of matching slot

Most matches when view = sourceView:

rcvr

aPoint

rcvr	x	3
rcvr	y	4
rcvr	*	return (rho
rcvr	rho	return ((x
rcvr	theta	return y arctan: x

Can get source code for any slot

Copyright © IBM Corporation, 2014

Different views can trigger/shape introspection

Possible Views on System (Introspections)	
name	what comes back from a matching slot
normal	result of method invocation
functional	access returns method object
slot	“slot mirror”
version	version of code in the slot
link	mirror reifying reference from slot to contents
provenance	Who wrote the code? Whence the data?
behavioral	“mirror” collapsing inheritance and super

Copyright © IBM Corporation, 2014

Different views can trigger/shape intercession

Possible Behaviors (Intercessions)	
name	behavior
base	business as usual
immutable	no assignments allowed
ensemble	ambiguous lookups run all methods
super	coordinate gives next place to look
objective	prohibits context changes
fleeting	context reverts on next send

Copyright © IBM Corporation, 2014

Conclusion

- Synergy of 3 ideas:
  - Slot-based model
  - Implicit arguments (context)
  - Multiple dispatch
- Simple, powerful model for dynamic contextual variation

Copyright © IBM Corporation, 2014

Thank you!

Thanks for encouragement / comments:

Sam Adams
Suparna Bhattacharya
Andrew Black
James Noble
Mark Wegman

Shriram Krishnamurthi (Onward! PC Chair)
Onward! Reviewers

Copyright © IBM Corporation, 2014