Conditional ACEs *and* Claims

how do they work and what are they good for?

Douglas Bagnall with a little help from Joseph Sutton

SAMBA dbagnall@samba.org

catalyst douglas.bagnall@catalyst.net.nz

Conditional ACEs

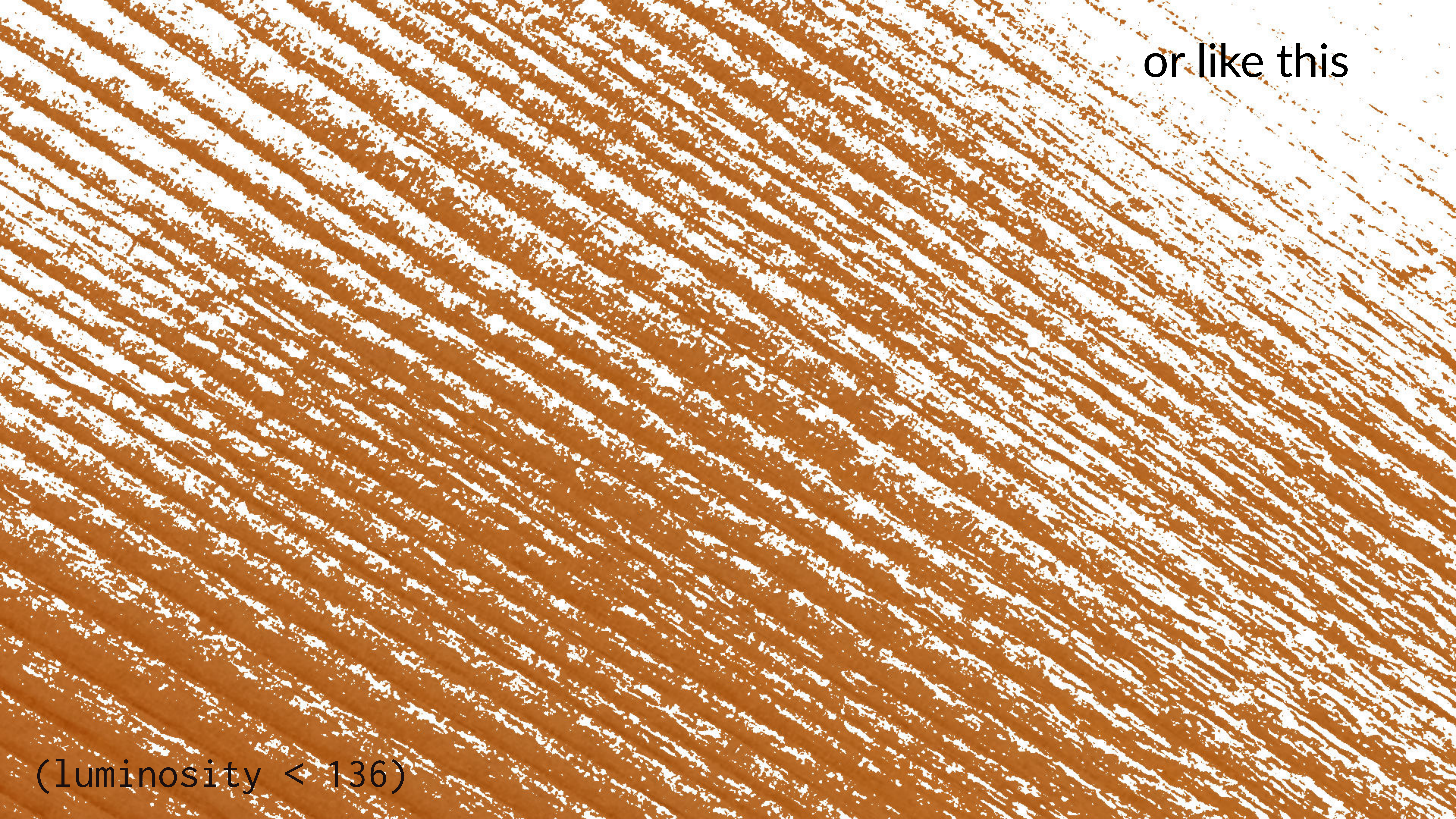
ACEs are Entries in an Access Control List

typically an ACE allows or denies specified access to a specified user, group or session

this allows fine-grained control, but grain is grain

easy to split like this

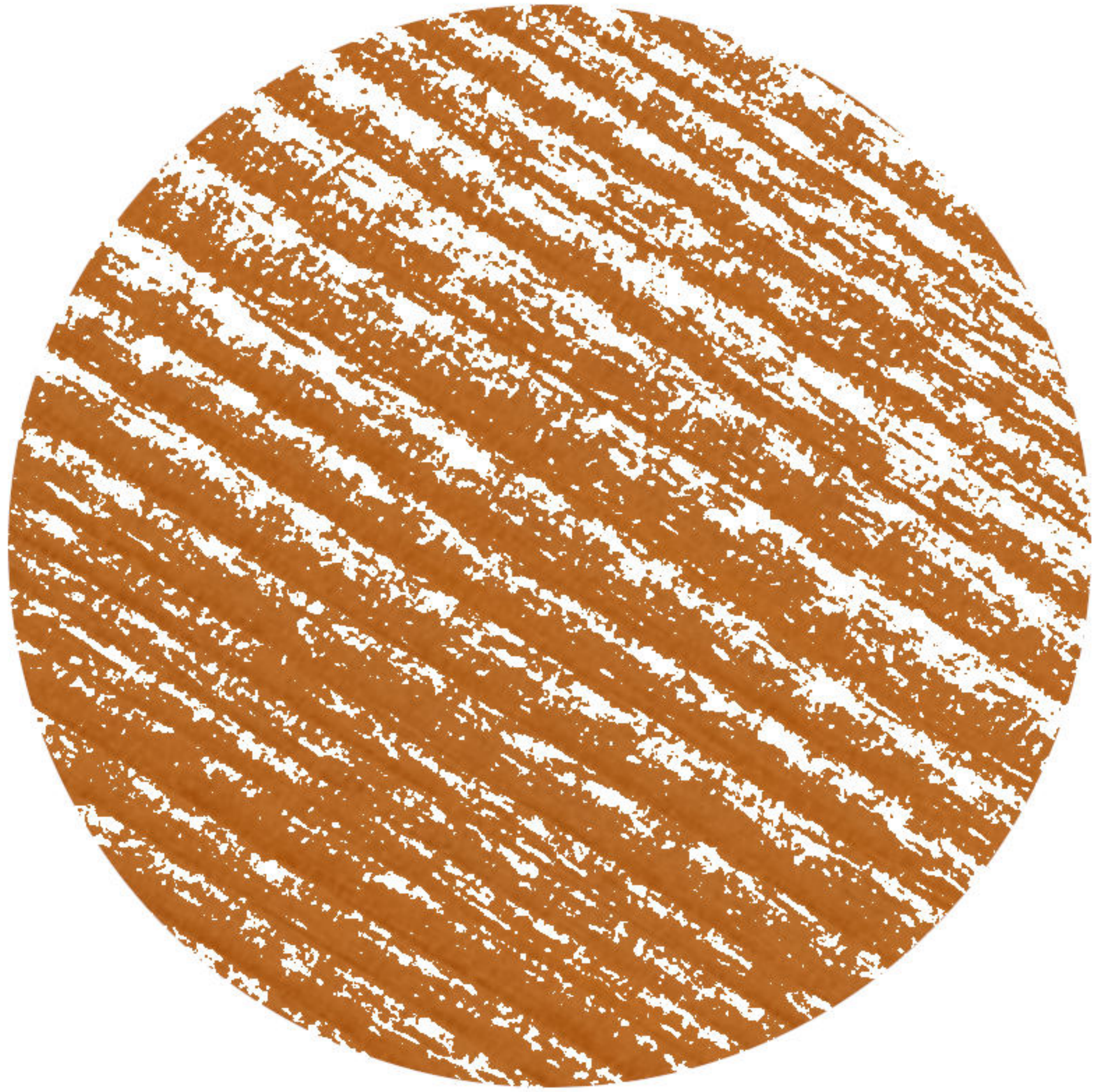
`(distance_from_x < 200)`



or like this

(luminosity < 136)

or like this



```
(luminosity < 136 && distance < 200)
```

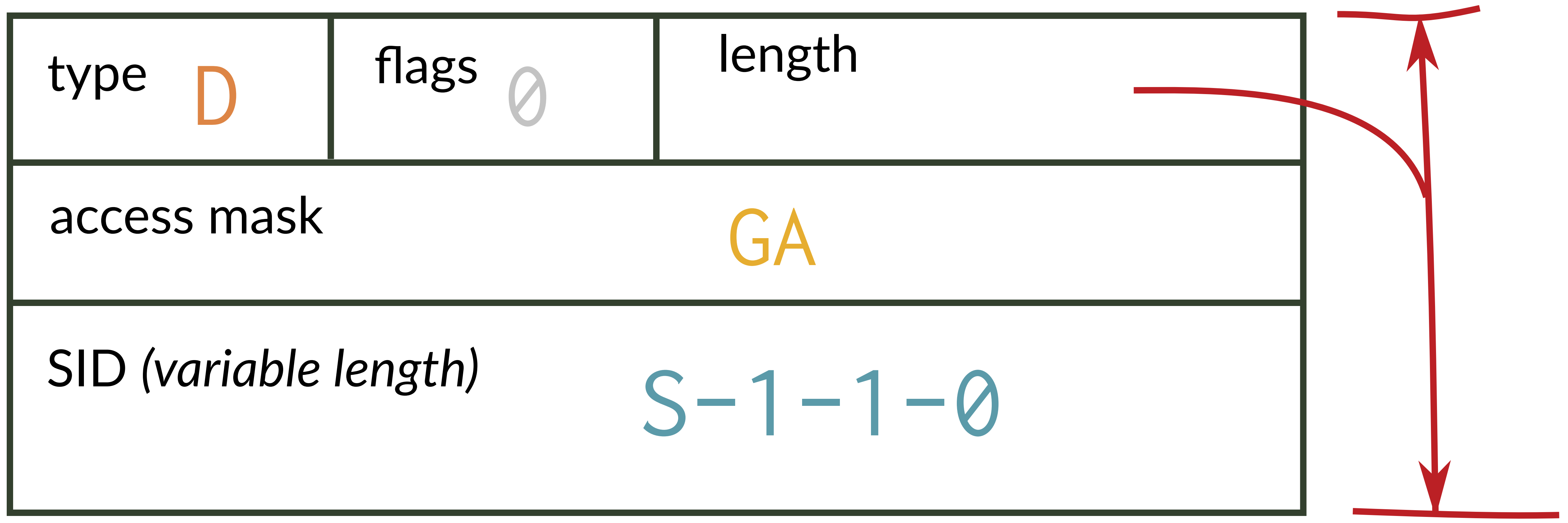
A conditional ACE is a “callback” ACE

XA	allow callback ACE
XD	deny callback ACE
XU	object allowed callback ACE
ZA	audit callback ACE

named for their implementation in Microsoft's AuthZ API

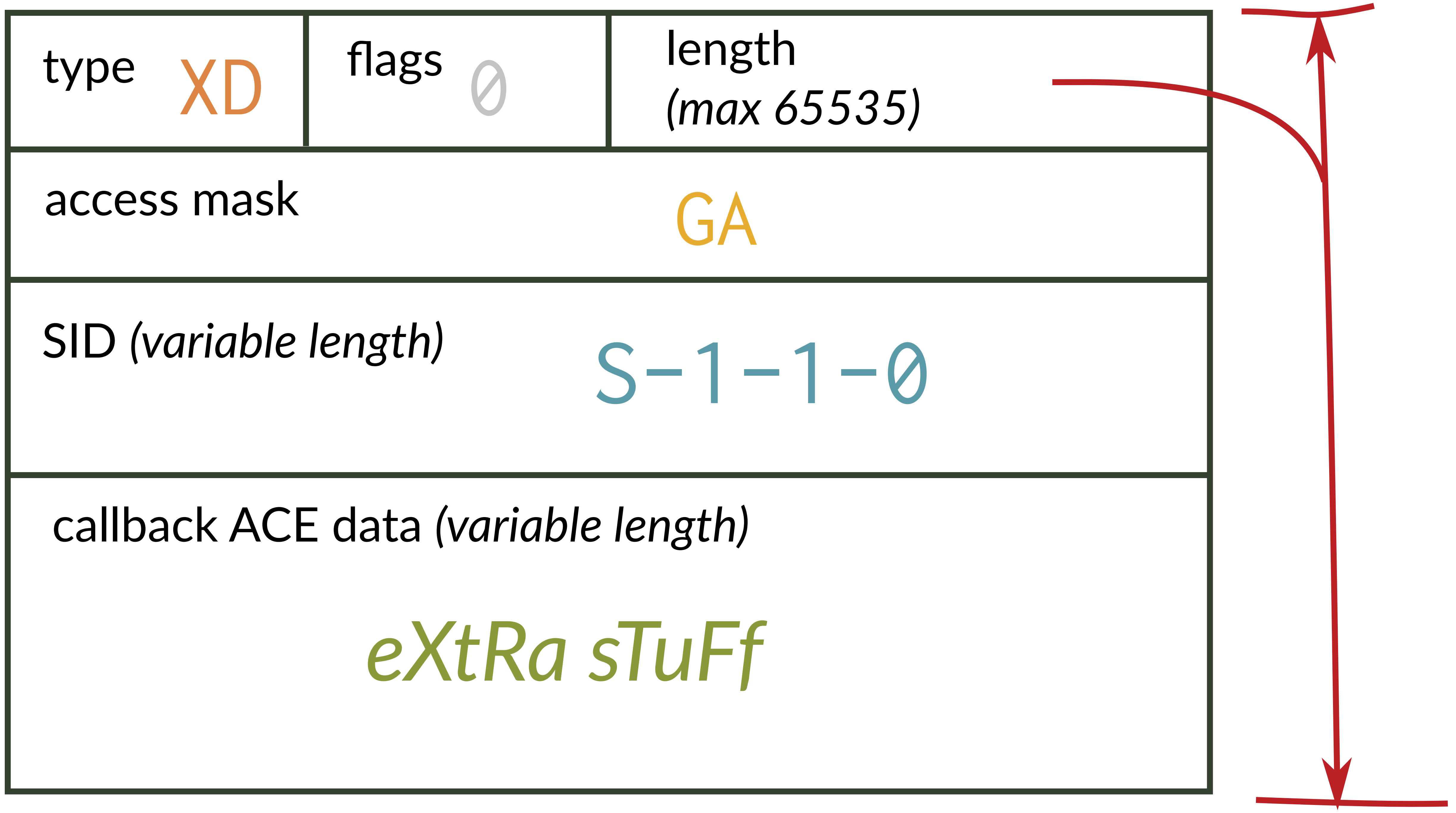
```
D:(D;;GA;;;S-1-1-0)
```

simple ACE structure



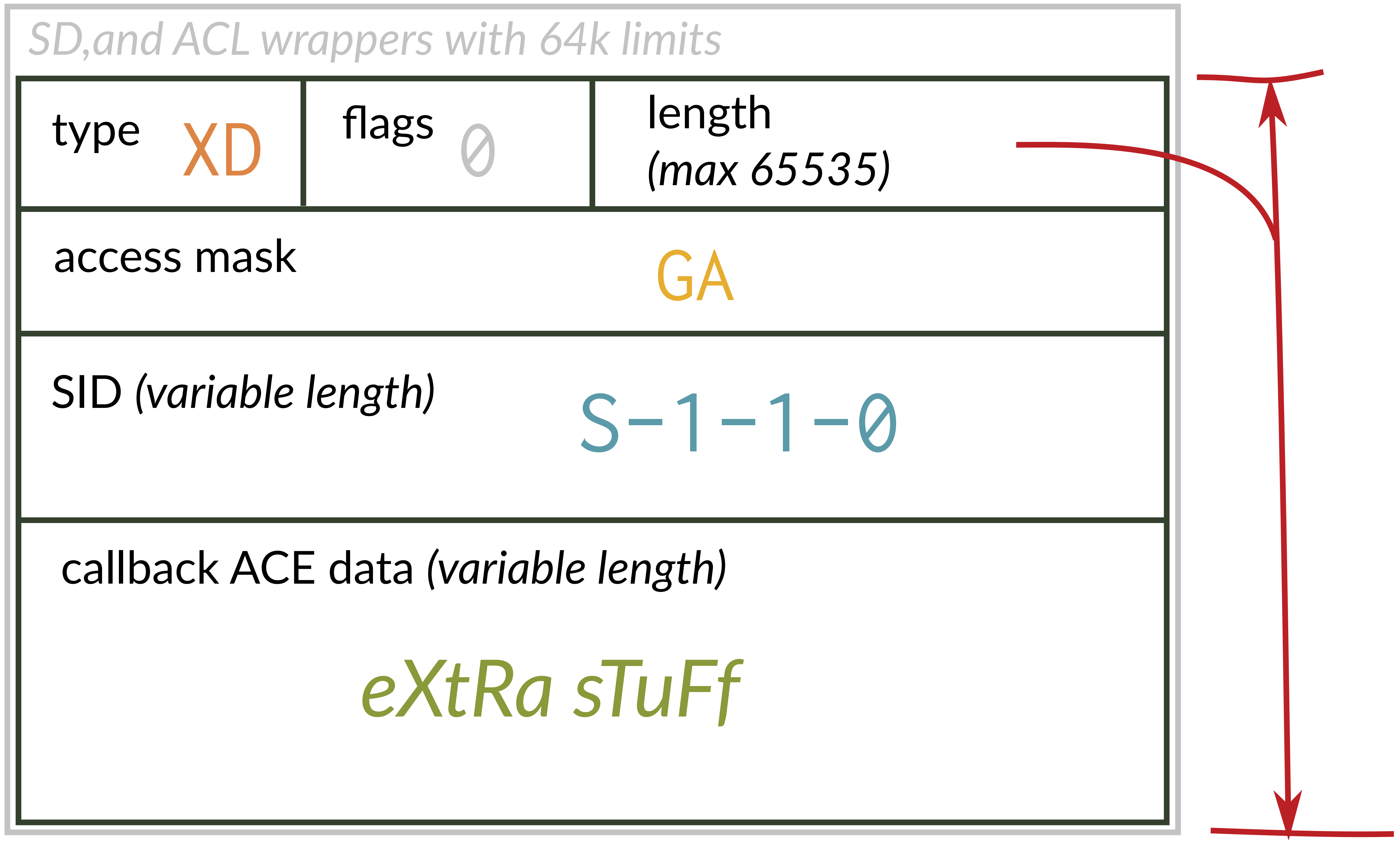
```
D:(XD;;GA;;;S-1-1-0;eXtRa sTuFf)
```

callback ACE structure


type XD
flags 0
length (max 65535)
access mask GA
SID (variable length) S-1-1-0
callback ACE data (variable length)
eXtRa sTuFf

`D:(XD;;GA;;;S-1-1-0;eXtRa sTuFf)`

callback ACE structure

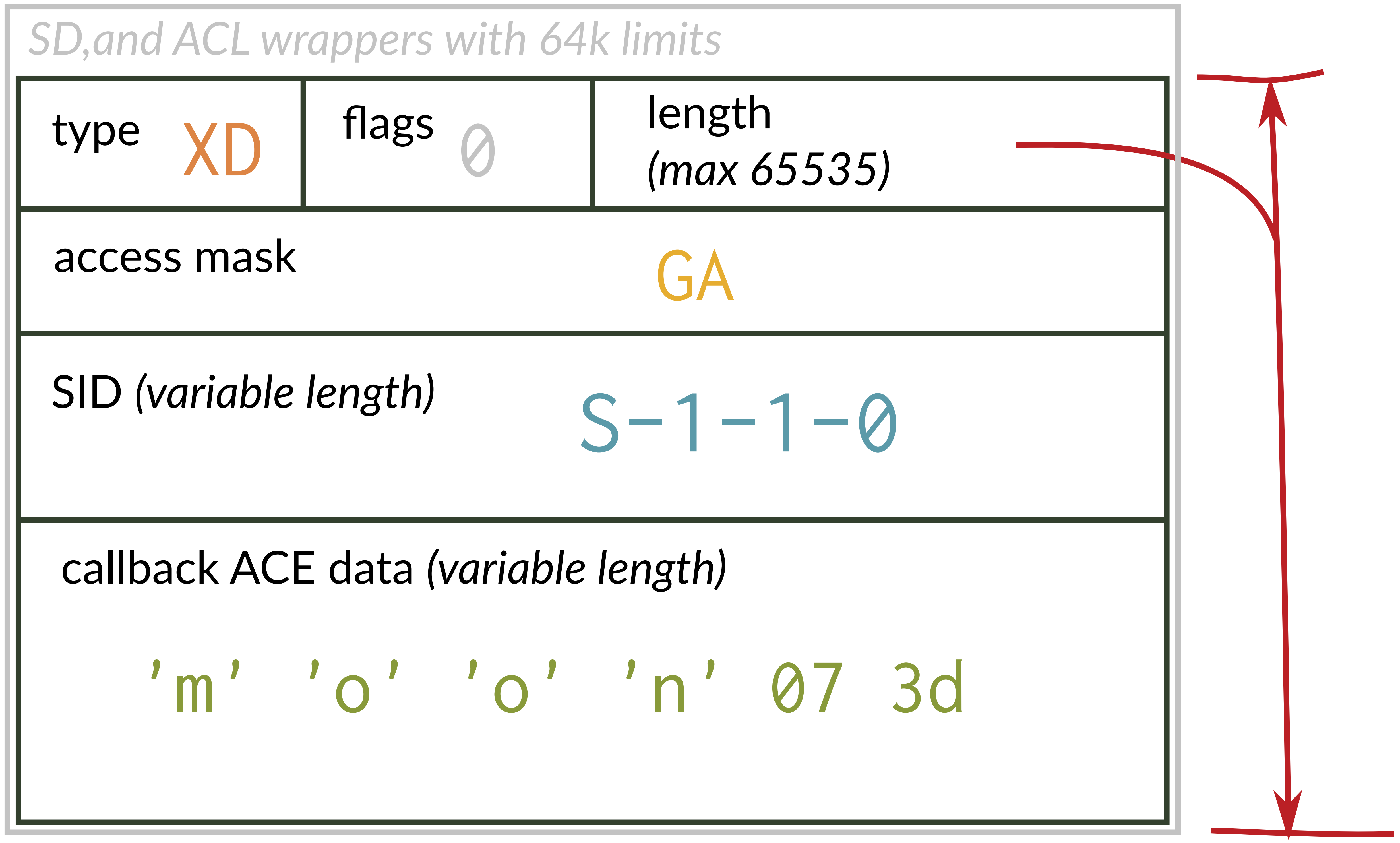


SD,and ACL wrappers with 64k limits
type XD
flags 0
length (max 65535)
access mask GA
SID (variable length) S-1-1-0
callback ACE data (variable length)
eXtRa sTuFf

D:(XD;;GA;;;WD;☾: waning gibbeous)

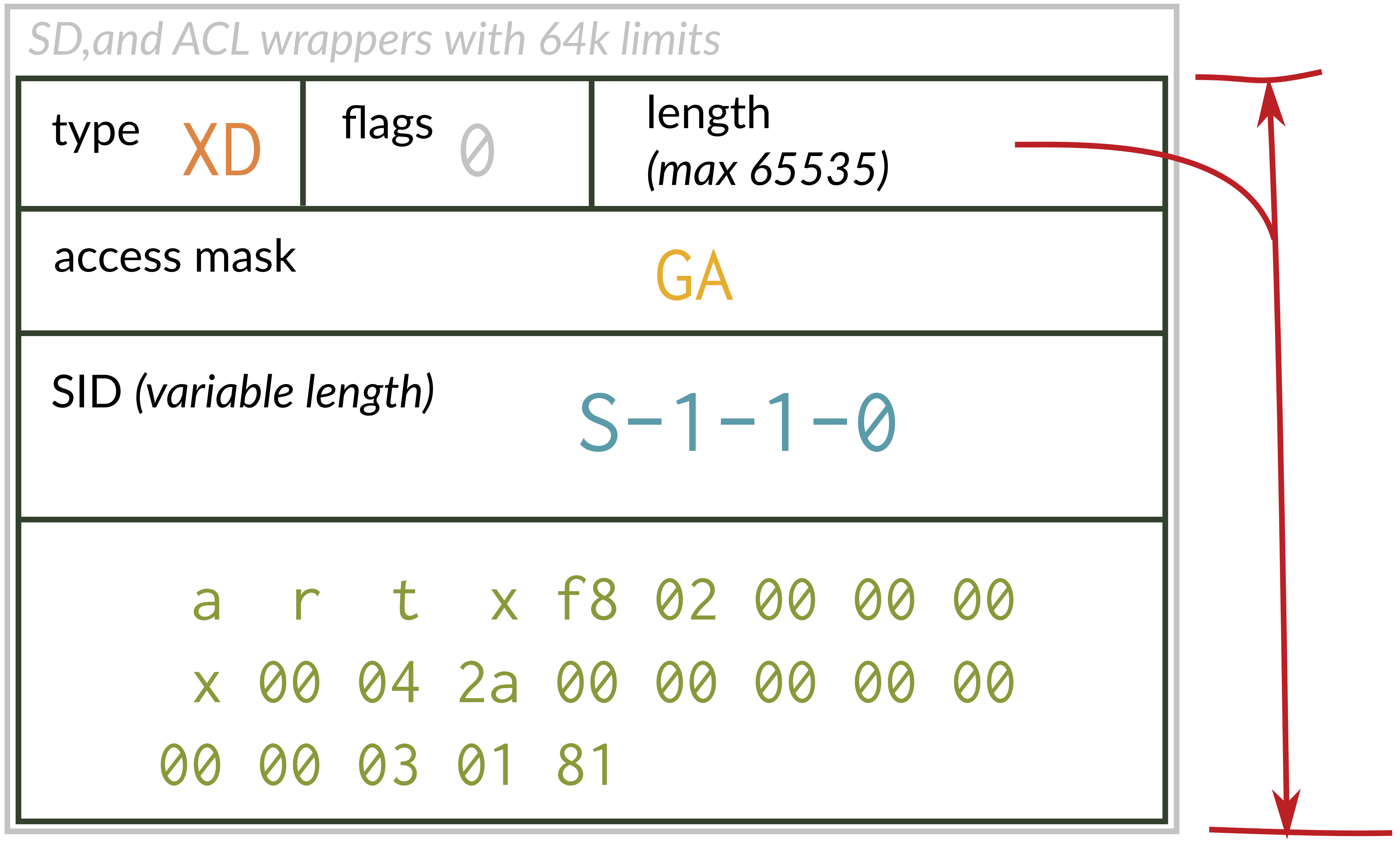
Windows has a mechanism for registering arbitrary callbacks

possibly completely unused, at least for allow and deny



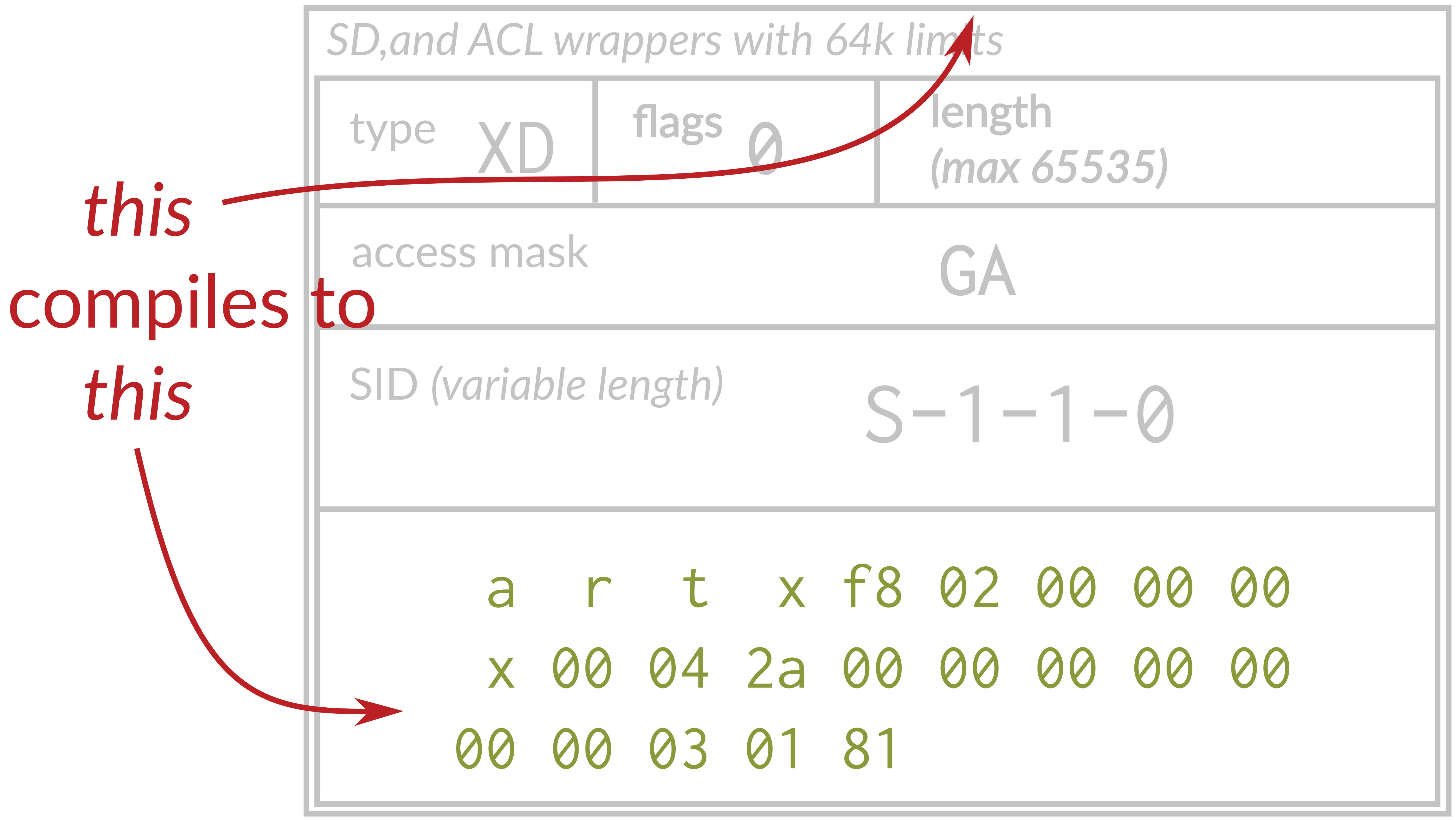
```
D:(XD;;GA;;;S-1-1-0;(x == 42))
```

conditional ACEs use “artx” magic prefix and ( ) markers in SDDL


SD,and ACL wrappers with 64k limits
type XD
flags 0
length
(max 65535)
access mask GA
SID (variable length) S-1-1-0

```
 a  r  t  x f8 02 00 00 00
 x 00 04 2a 00 00 00 00 00
00 00 03 01 81
```

```
D:(XD;;GA;;;S-1-1-0;(x == 42))
```



```
 a  r  t  x f8 02 00 00 00
 x 00 04 2a 00 00 00 00 00
00 00 03 01 81
```

(x == 42)

`a r t x`	*magic number*
`f8`	*local attribute*
`02 00 00 00`	*length of name (in bytes)*
`x 00`	*name (utf-16)*
`04`	*int64*
`2a 00 00 00 00 00 00 00`	*value (42)*
`03`	*display integer sign (none)*
`02`	*display integer base (decimal)*
`81`	*equality operator*

Conditional ACE ternary logic

there is an unknown type
(works as you might expect)

AND	true	false	unknown
true	T	F	?
false	F	F	F
unknown	?	F	?

OR	true	false	unknown
true	T	T	T
false	T	F	?
unknown	T	?	?

NOT	
true	F
false	T
unknown	?

Conditional ACE ternary logic for `(x == 42)`

if there is no local attribute “x”, the result is *unknown*

if local *x* is not an integer, the result is *unknown*

if this is an *XD* ACE, unknown means *yes*, *deny*

if this is an *XA* ACE, unknown means *no*, *do not allow*

Conditional ACE attributes

what is this “local attribute” and
where did it come from?

put that thought aside for the moment.

Conditional ACE examples

```
D:(XD;;FA;;;S-1-1-0;(@User.Title == "PM"))
```

meaning: users with the title “PM”
are *not* allowed to access this

```
(@User.Title=="PM" && (@User.Division=="Finance" || @User.Division =="Sales")
```

meaning: selects users with the title “PM”
from the “Finance” or “Sales” divisions

Conditional ACE examples

```
D:(XA;;FR;;;WD;(Member_of {SID(S-1-234-56), SID(BO)} && @Device.Bitlocker))
```

allows users who are members of *both* these SIDs if the device attribute “Bitlocker” is also true.

```
D:(XA;;FX;;;S-1-1-0; (@User.Project Any_of @Resource.Project))
```

allows users whose “Project” attribute is in the resource attribute “Project” (which is presumed to be a list of 1 or more values).

Conditional ACE examples

```
O:SYG:SYD:(XA;OICI;CR;;;WD;(@USER.ad://ext/AuthenticationSilo == "tier 0"))
```

“@USER.ad://ext/AuthenticationSilo” is a computed attribute and part of how authentication silos work.

This is allowing access to those users computed to be in the “tier 0” silo.

```
(@User.clearanceLevel >= @Resource.requiredClearance))
```

Maybe this user is a spy

Conditional ACE operators

`>   >=   ==   <=   <   &&   ||   !`

`Member_of`	`Not_Member_of`
`Member_of_Any`	`Not_Member_of_Any`
`Device_Member_of`	`Not_Device_Member_of`
`Device_Member_of_Any`	`Not_Device_Member_of_Any`
`Contains`	`Not_Contains`
`Any_of`	`Not_Any_of`
`Exists`	`Not_Exists`

composite list constructor `{ }`　　　　logical grouping `( )`

Conditional ACE types

`int64 int32 int16 int8`	*only int64 can be expressed in SDDL; have flags for sign and base display hints*
`Unicode string`	`"hello"`
`octet string`	`#68656c6c6f0a`
`composite`	`{1, 2, {3, "four"}, SID(BA)}`
`SID`	`SID(S-1-2-3)`
`result type`	*true, false, or unknown; true or false can be expresed as 1 and 0*

Conditional ACE attributes

`@User.attr`	claims issued to the user
`@Device.attr`	claims issued to the user's computer
`@Resource.attr`	from a *Resource Attribute ACE* in the accessed thing's SACL.
`attr`	“local” claims issued to the authenticated principal

Syntactically, in SDDL, local attributes are restricted to ASCII-word-ish strings

Resource Attribute ACE

```
(RA;CI;;;;WD; ("Project",TS,0,"Samba","Heimdal"))
(RA;CI;;;;S-1-1-0; ("requiredClearance",TU,0,3))
```

```
(RA;flags;;;;WD;(name, type, flags, value))
```

TI	*signed 64-bit integer*	TD	*SID string*
TU	*unsigned 64 bit integer*	TX	*octet string*
TS	*unicode string*	TB	*boolean value (1|0)*

note: types don't line up exactly with Conditional ACE types

Resource attributes are from another new ACE type

They hide in SACLs

accessed via the *@Resource*. conditional ACE syntax

these examples are the same:

```
  D:(XA;;;;WD;(@User.foo == 1))

  D:(XA;;;;WD;(@User.foo == @Resource.foo))
  S:(RA;;;;WD;("foo",TI,0,1))
```

but the conditional ACE could be inherited

User claims, device claims, local claims

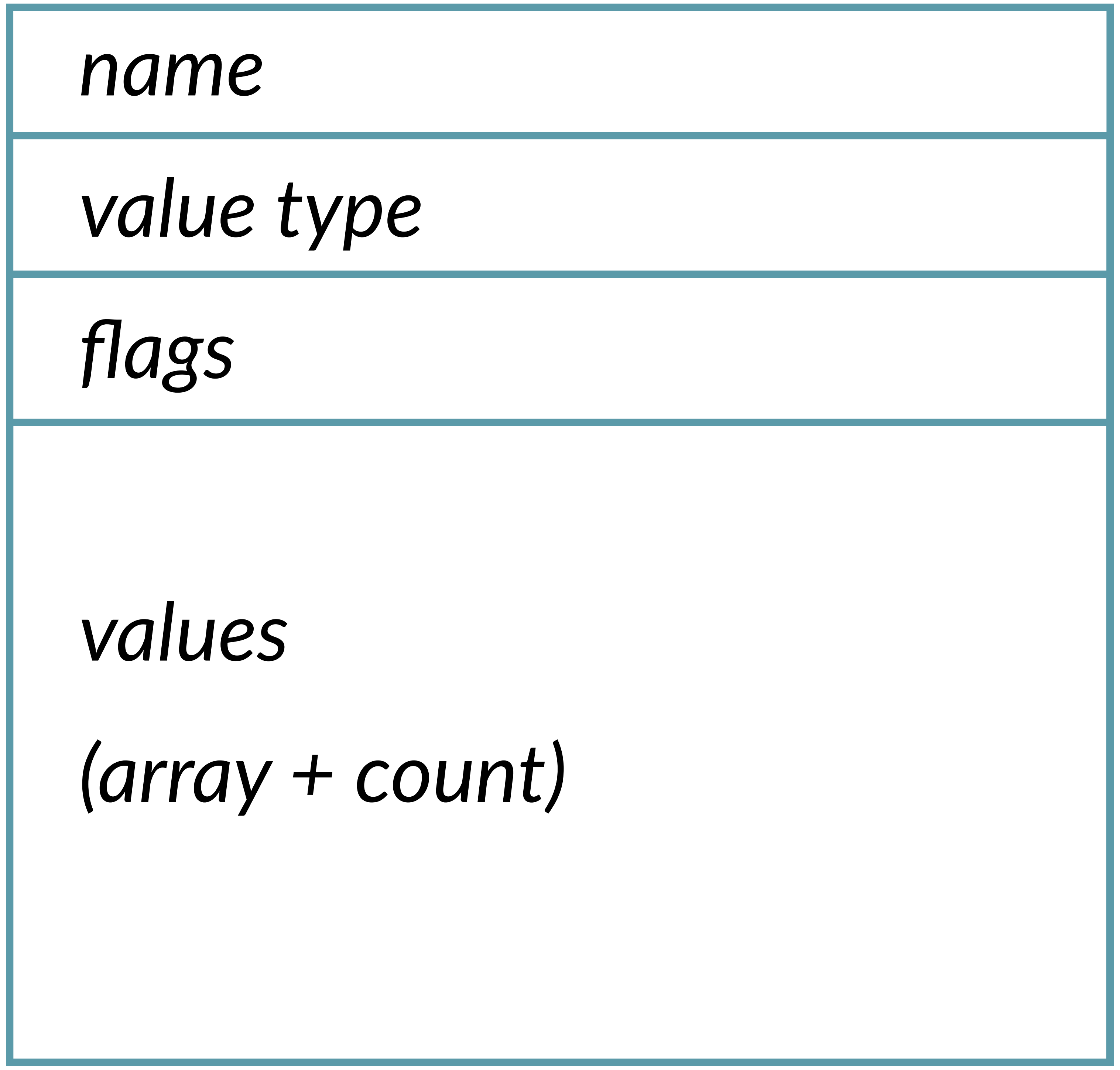
from the ACE's point of view these come from the security token.

```
{SID, SID, SID,…},
privilege mask,
rights mask,
{user claim, user claim, user claim,…},
{device claim, device claim,…},
{local claim, local claim, local claim,…},
{device SID, device SID,…},
```

security tokens have these

and need to have these

A Claim object



same types as resource ACE:

int64

uint64

unicode string

SID string

boolean

byte string

Security token claims come from the PAC

A kerberos ticket can contain a PAC;
a PAC can contain “claims blobs”.

PAC claims seem to have different types again
(no SIDs, octet strings).

The PAC claims come from the KDC.

The KDC looks stuff up in ldb.

What are claims, really?

A snapshot of values from the database that float off with a kerberos ticket, cleverly wrapped so that things using the ticket can trust the claims and don't need to pester the database.

So things on the edge can make complex secure authorization decisions, without database access.

Conditional ACEs without Kerberos?

It could work if you can ask a DC

very slowly

otherwise conditions resolve to *unknown*
(*deny* for deny ACEs, *not allow* for allow ACEs)

wherefore claims and condtional ACEs?

Claims enable secure decentralisation of complex authorization decisions

Conditional ACEs are the mechanism used

Authentication silos involve magic computed claims

2012R2 functional levels

Questions?

(ask Joseph)