

# CTHULHU

Using Tcl to build C to build Tcl Applications

# CTHULHU

**C** and **T**cl **H**ashtable **U**nderpinnings for  
**L**anguage **H**igh-level **U**nderstanding

# CTHULHU

**C** and **Tcl** **H**ashtable **U**nderpinnings for  
**L**anguage **H**igh-level **U**nderstanding

CTH

C  
a  
L



hello cthulhu

# CTHULHU

**C** and **Tcl** **H**ashtable **U**nderpinnings for  
**L**anguage **H**igh-level **U**nderstanding



# Background

# Background

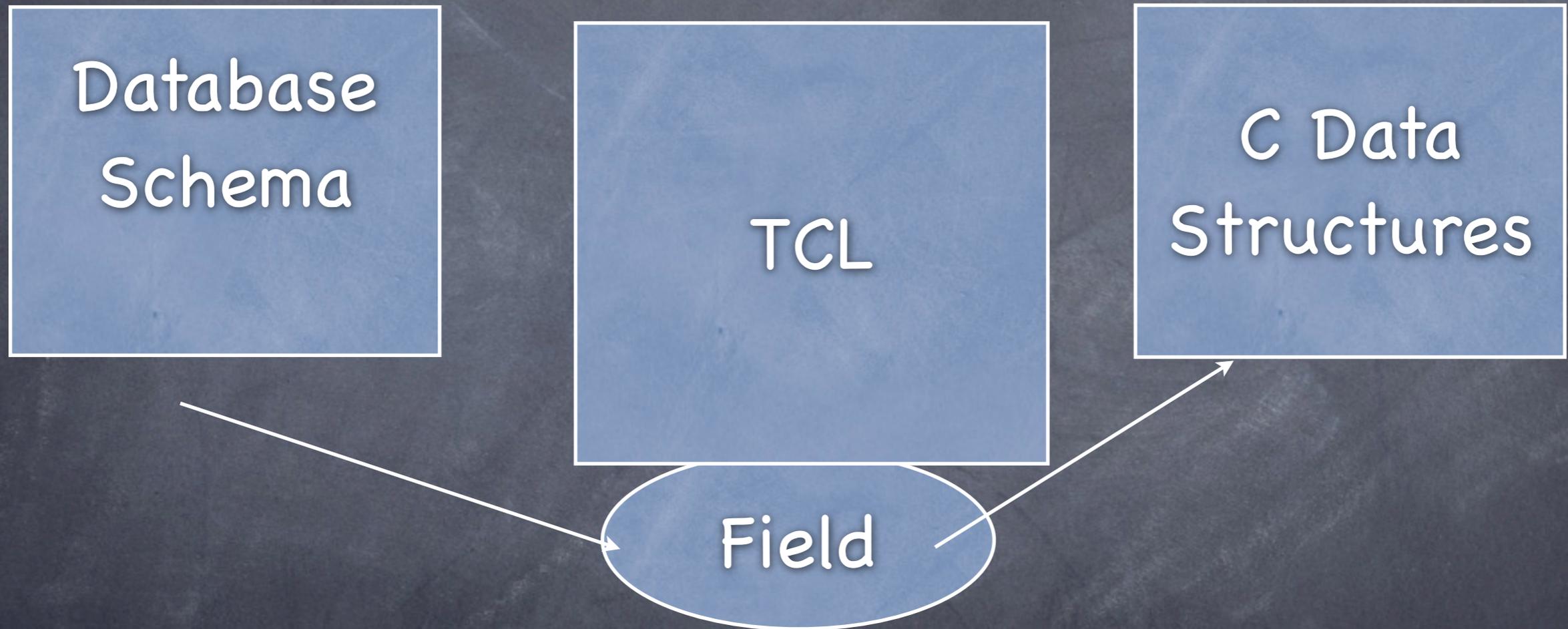
Database  
Schema

# Background

Database  
Schema

C Data  
Structures

# Background



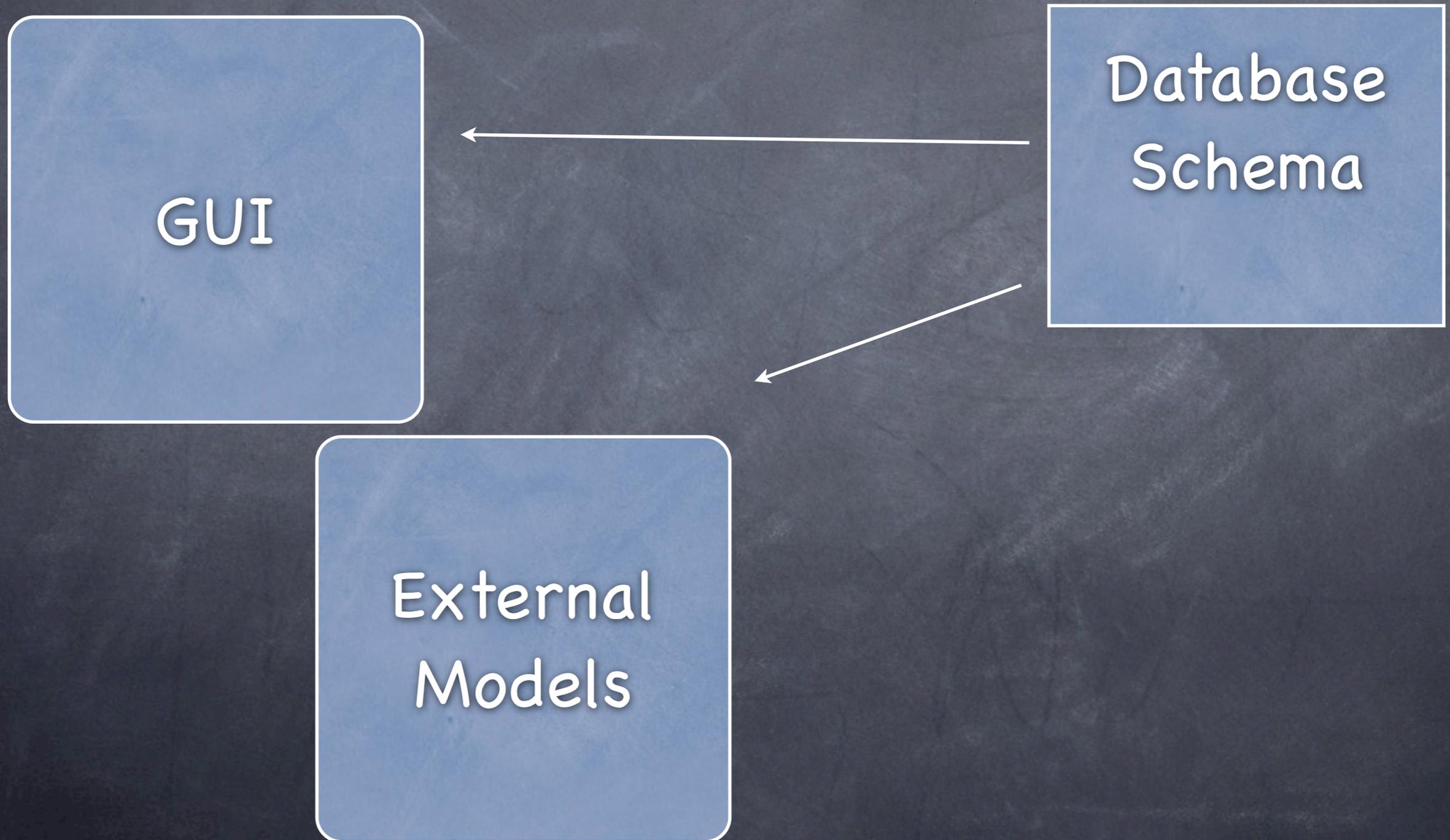
# Background

Database  
Schema

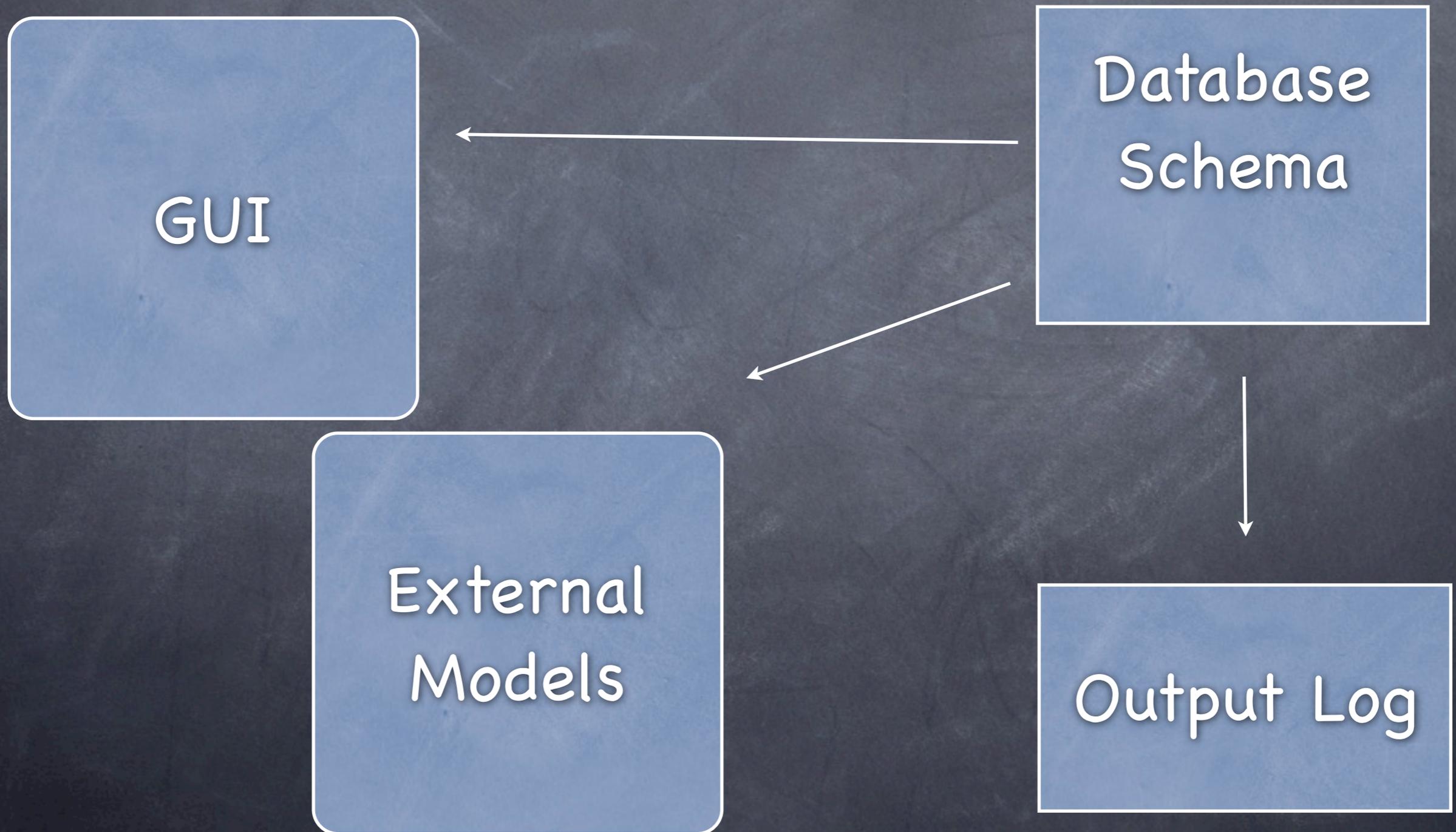
# Background



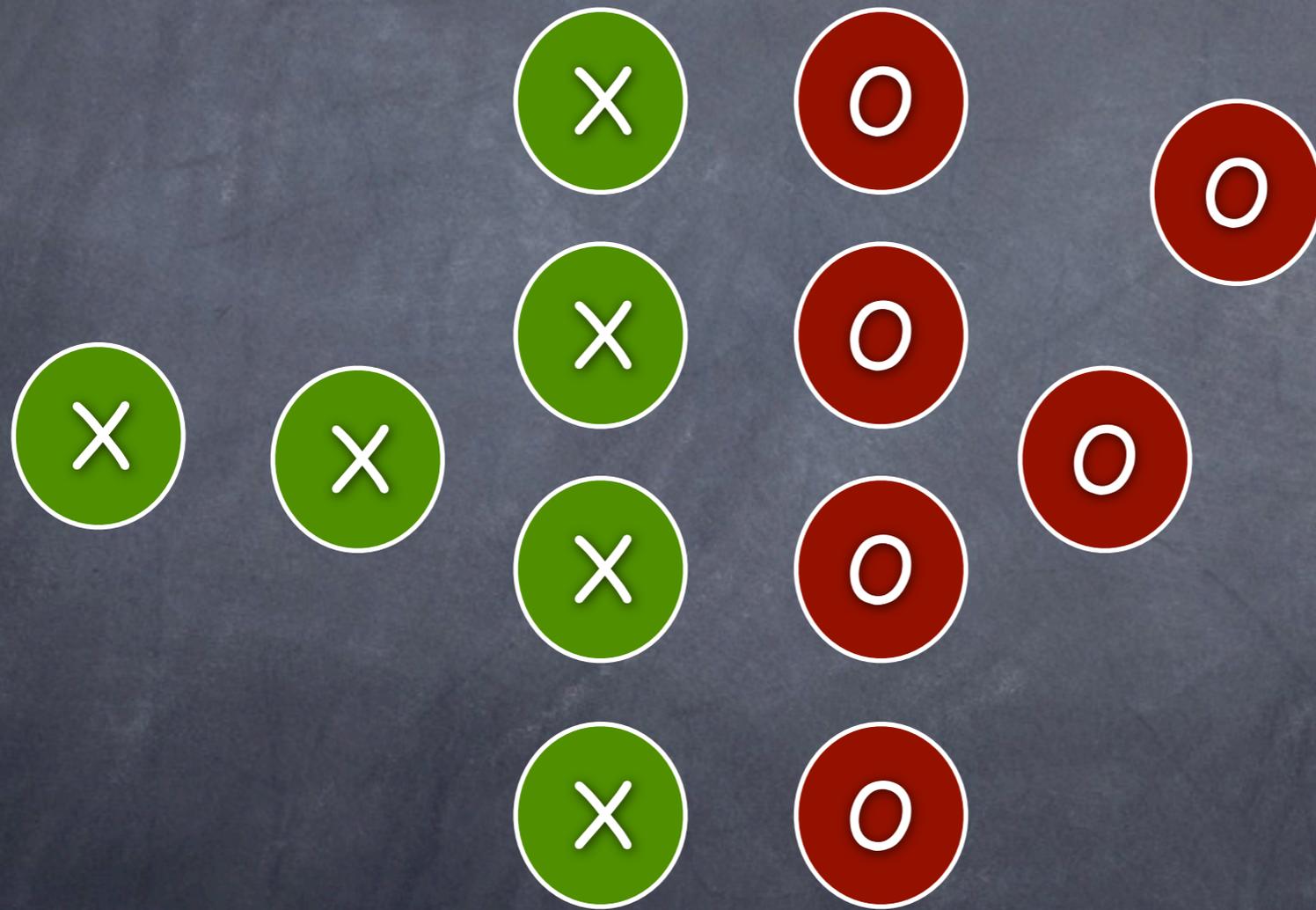
# Background



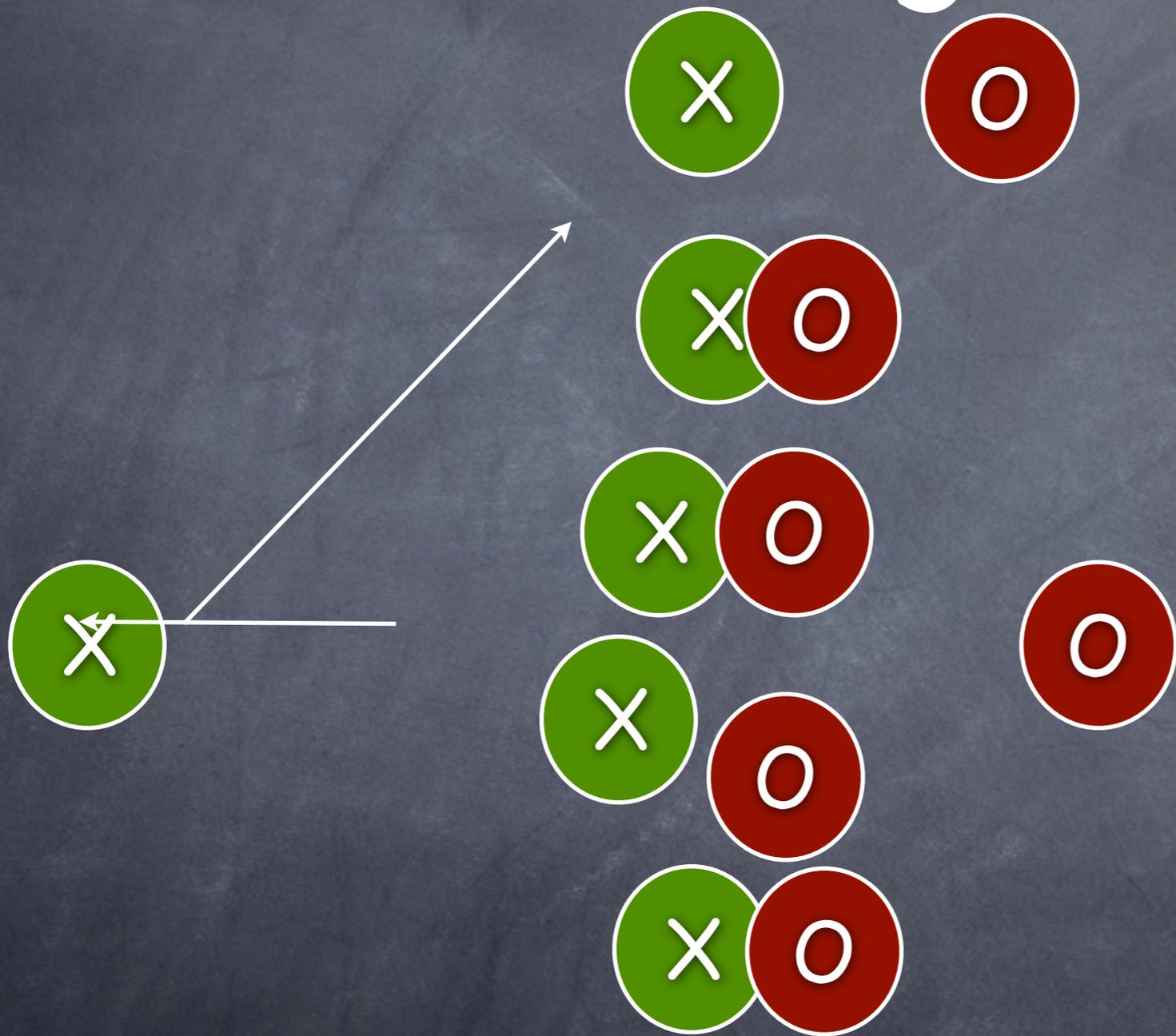
# Background



# Background



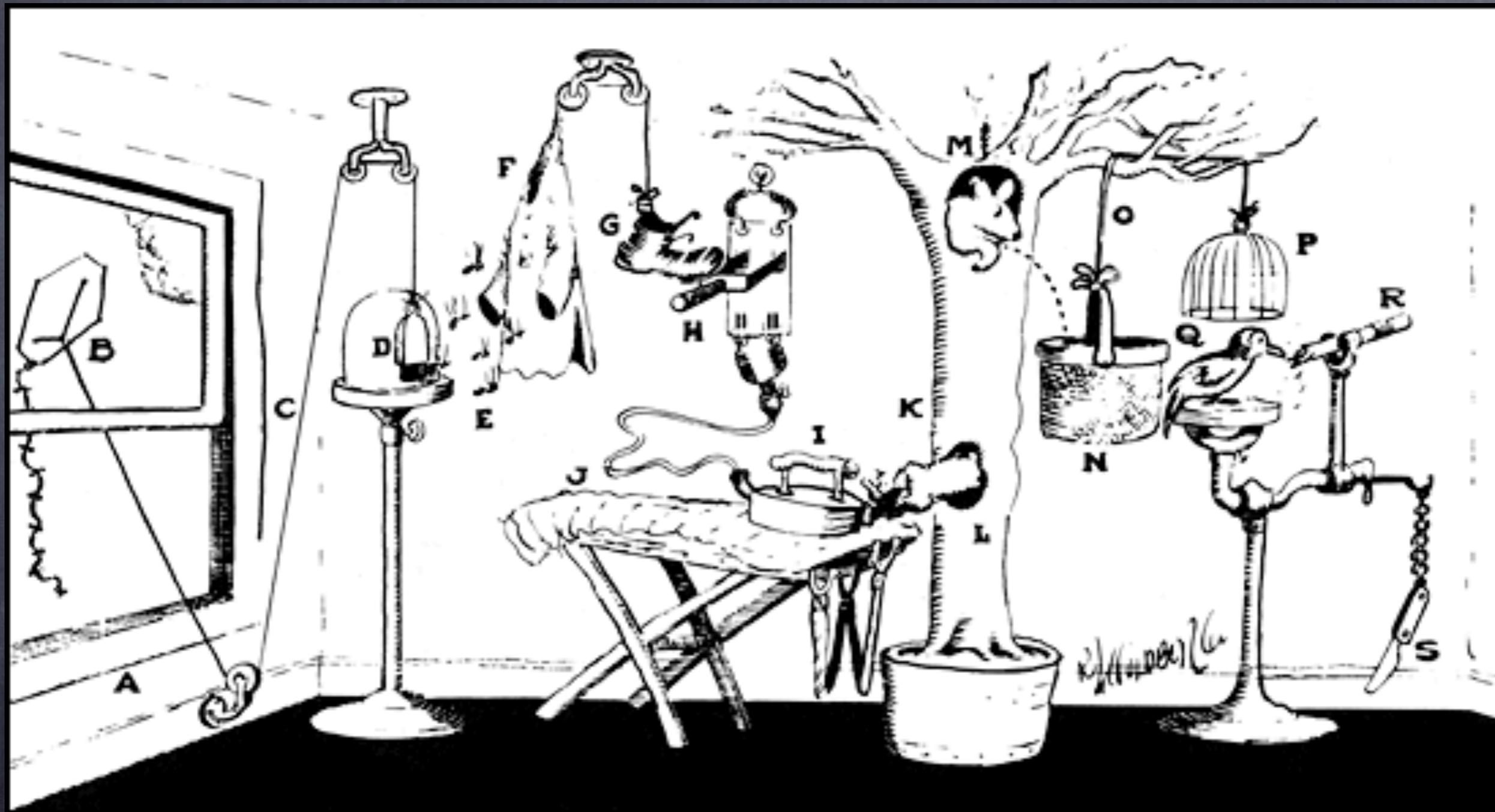
# Background







# Background



# Description Language

Description  
Language



SQL

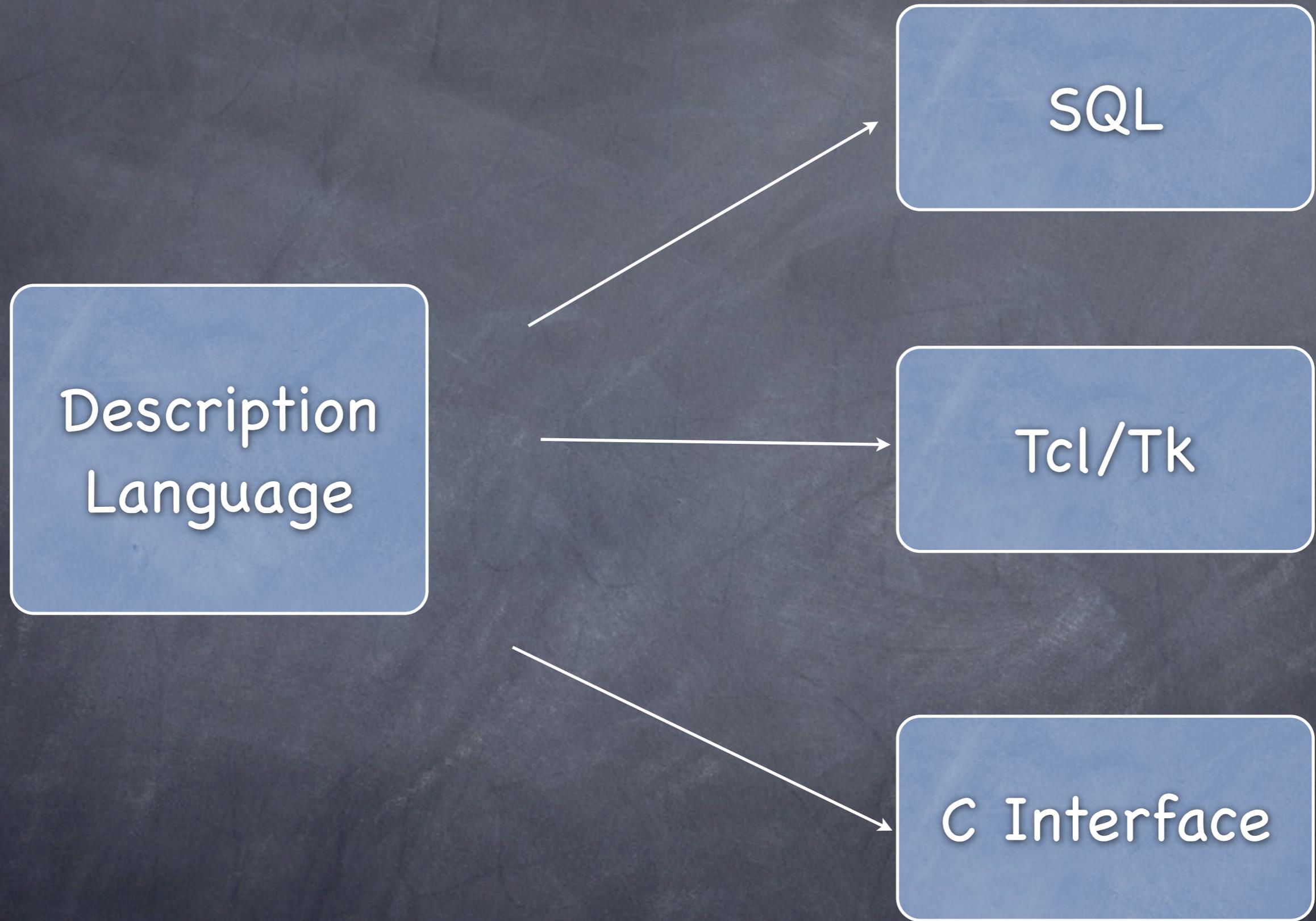
Description  
Language

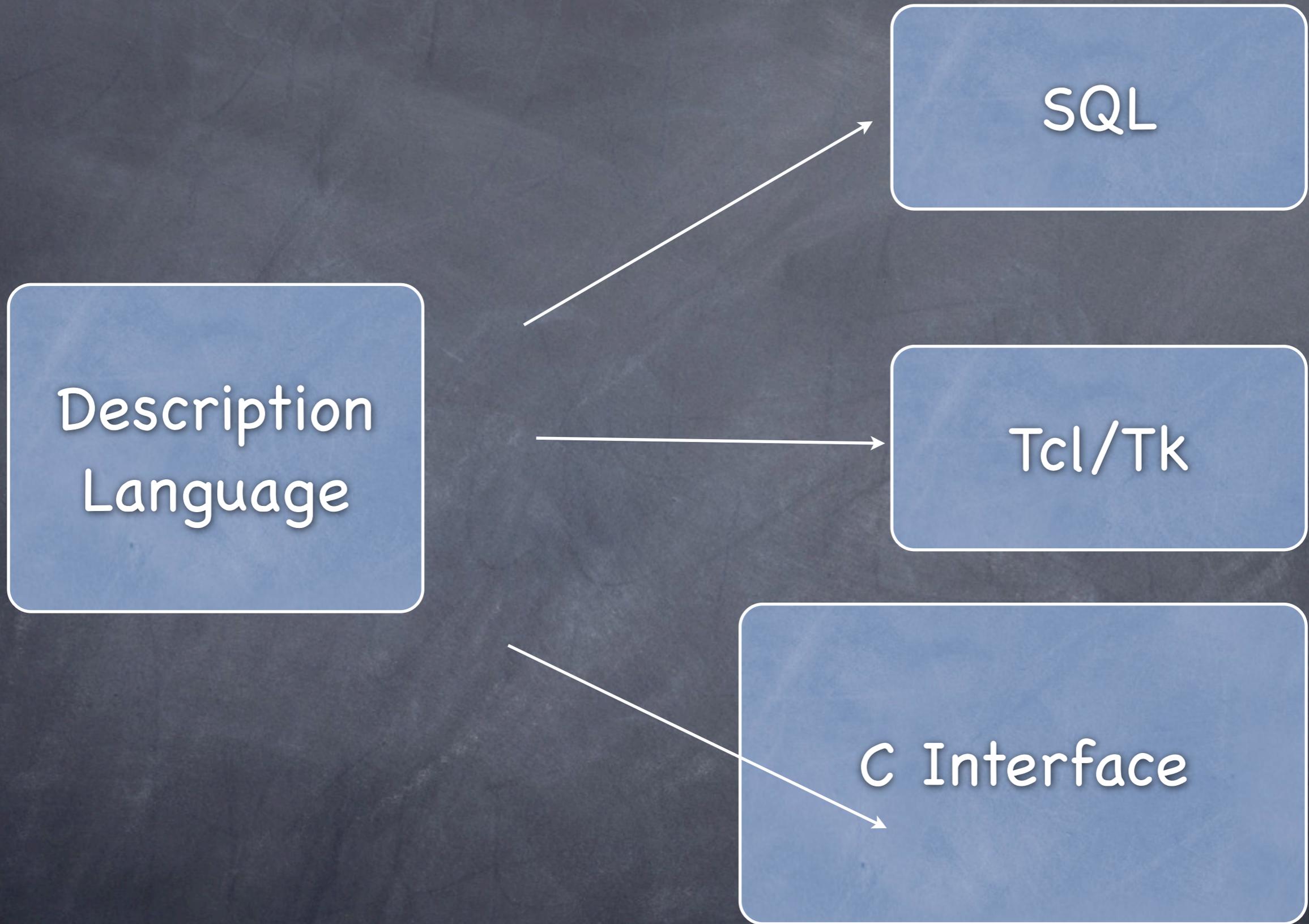


SQL



Tcl/TK





Description  
Language

C Interface

# CTHULHU Description Language

- Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn

# CTHULHU Description Language

# CTHULHU Description Language

- Dict Based

# CTHULHU Description Language

- Dict Based
- Combines the needs of the GUI, the database, and C

# CTHULHU Description Language

- Dict Based
- Combines the needs of the GUI, the database, and C
- Captures suggested storage, display, validation rules

# CTHULHU Description Language

- Dict Based
- Combines the needs of the GUI, the database, and C
- Captures suggested storage, display, validation rules
- Comes with a free frogurt

# CTHULHU Description Language

```
cstructDefine portal {
```

```
}
```

# CTHULHU Description Language

```
cstructDefine portal {  
    keytype int  
}
```

# CTHULHU Description Language

```
cstructDefine portal {  
    keytype int  
    hashtable tcl  
}
```

# CTHULHU Description Language

```
cstructDefine portal {  
  keytype int  
  hashtype tcl  
  fields {  
  }  
}
```

```
cstructDefine portal {  
  keytype int  
  hashtype tcl  
  fields {  
    toid { ... }  
    fromid { ... }  
  }  
}
```

...

```
toid {  
  storage Roid  
  references compt  
  description { ... }  
}
```

...

...

```
cake {  
  storage Bool  
  description {There will be cake}  
  default 1  
}
```

...

...

```
open {  
  storage u2  
  values {  
    0 closed {Closed}  
    1 open   {Open}  
    2 closing {Closing}  
    3 opening {Opening}  
  }  
}
```

...

```
cstructDefine portal {  
    keytype int  
    hashtype tcl  
    fields { ... }  
    static {  
        Roid id;  
        Entity *pType;  
        Portal *delta  
    }  
}
```

```

/* Define the Portal Structure */
#define PORTAL_JAMMED_NORMAL 0 /* Door...
#define PORTAL_JAMMED_JAM_OPEN 2 /* Do...
#define PORTAL_OPEN_NORMAL 0 /* Door i...
#define PORTAL_OPEN_OPEN 1 /* Door is ...
#define PORTAL_OPEN_OPENING 2 /* Crew ...
#define PORTAL_OPEN_CLOSING 3 /* Crew ...
struct Portal {
    /* Place holder for ID */
    Tcl_Obj *idString;
    /* Begin hard-coded structure */

    Roid id;          /* Numeric ID ...
    Entity *pType;
    Portal *delta;

    Compartment *pFrom; /* Compartment ...
    Compartment *pTo;   /* Compartment ...
    Crew *holding;     /* Crewmember ...
    Portal *airlock;   /* Pointer to i...

    float public_flooding; /* flooding e...
    Roid public_fromid; /* fromid eoc */
    Roid public_toid; /* toid eoc */
    /*Begin bitmap*/
    unsigned int public_jammed :4; /*...
    unsigned int public_open :4; /* o...
    unsigned int public_reserved :1; ...
};

```

```

const struct IrmParamNameMap Portal_paramNameMap[] = {
    { "damage", CSTRUCT_PORTAL_DAMAGE, PTYPE_INT },
    { "flooding", CSTRUCT_PORTAL_FLOODING, PTYPE_FLOAT },
    { "fromid", CSTRUCT_PORTAL_FROMID, PTYPE_INT },
    { "junction_type", CSTRUCT_PORTAL_JUNCTION_TYPE, PTYPE_INT },
    { "open", CSTRUCT_PORTAL_OPEN, PTYPE_INT },
    { "opening", CSTRUCT_PORTAL_OPENING, PTYPE_FLOAT },
    { "reserved", CSTRUCT_PORTAL_RESERVED, PTYPE_INT },
    { "toid", CSTRUCT_PORTAL_TOID, PTYPE_INT },
};

void Portal_DictPut(Portal *p, Tcl_Obj *key, Tcl_Obj *value) {
    if(!p->pType) {
        Portal_Generic_Alloc(p);
    }
    Entity_DictPut(p->pType, key, value);
}

Tcl_Obj *Portal_DictGet(Portal *p, Tcl_Obj* key) {
    if(!p->pType) {
        return NULL;
    }
    return Entity_DictGet(p->pType, key);
}

/* Define the $StructName_StructSet function */

int Portal_StructSet(Tcl_Interp *interp, Portal *pNode, int field, Tcl_Obj *value)
{
    if(field < 0 || field > CSTRUCT_PORTAL_Count) {
        const struct IrmParamNameMap *aParam = Spec_paramNameMap;
        Tcl_AppendResult(interp, aParam[field].zName, " is not a field for $structname", 0);
        return TCL_ERROR;
    }

    switch (field) {
        case CSTRUCT_PORTAL_DAMAGE: {
            int intValue;
            double floatValue;
            if(Tcl_GetDoubleFromObj(interp, value, &floatValue) == TCL_OK) {
                intValue = (int)floatValue;
            } else {
                /* Bag, we have a bad value */
                return TCL_ERROR;
            }
            pNode->public_damage = intValue; return TCL_OK;
        }
        case CSTRUCT_PORTAL_FLOODING: {
            double floatValue;
            if(Tcl_GetDoubleFromObj(interp, value, &floatValue) == TCL_OK) {
                pNode->public_flooding = (float)floatValue; return TCL_OK;
            }
            return TCL_ERROR;
        }
        case CSTRUCT_PORTAL_FROMID: {
            int intValue;
            double floatValue;
            if(Tcl_GetDoubleFromObj(interp, value, &floatValue) == TCL_OK) {
                intValue = (int)floatValue;
            } else {
                /* Bag, we have a bad value */
                return TCL_ERROR;
            }
        }
    }
}

```

```

        intValue=(int)floatValue;
    } else {
        /* Bag, we have a bad value */
        return TCL_ERROR;
    }
    pNode->public_fromid=intValue; return TCL_OK;
}
case CSTRUCT_PORTAL_JUNCTION_TYPE: {
    int intValue;
    double floatValue;
    if(Tcl_GetDoubleFromObj(interp,value,&floatValue)==TCL_OK) {
        intValue=(int)floatValue;
    } else {
        /* Bag, we have a bad value */
        return TCL_ERROR;
    }
    pNode->public_junction_type=intValue; return TCL_OK;
}
case CSTRUCT_PORTAL_OPEN: {
    int intValue;
    double floatValue;
    if(Tcl_GetDoubleFromObj(interp,value,&floatValue)==TCL_OK) {
        intValue=(int)floatValue;
    } else {
        /* Bag, we have a bad value */
        return TCL_ERROR;
    }
    pNode->public_open=intValue; return TCL_OK;
}
case CSTRUCT_PORTAL_OPENING: {
    double floatValue;
    if(Tcl_GetDoubleFromObj(interp,value,&floatValue)==TCL_OK) {
        pNode->public_opening=(float)floatValue; return TCL_OK;
    }
    return TCL_ERROR;
}
case CSTRUCT_PORTAL_RESERVED: {
    int intValue;
    double floatValue;
    if(Tcl_GetDoubleFromObj(interp,value,&floatValue)==TCL_OK) {
        intValue=(int)floatValue;
    } else {
        /* Bag, we have a bad value */
        return TCL_ERROR;
    }
    pNode->public_reserved=intValue > 0; return TCL_OK;
}
case CSTRUCT_PORTAL_TOID: {
    int intValue;
    double floatValue;
    if(Tcl_GetDoubleFromObj(interp,value,&floatValue)==TCL_OK) {
        intValue=(int)floatValue;
    } else {
        /* Bag, we have a bad value */
        return TCL_ERROR;
    }
    pNode->public_toid=intValue; return TCL_OK;
}
}
return TCL_OK;
}

```

```

/* Define the $StructName_StructGet function */

Tcl_Obj *Portal_StructGet(Portal *pNode,int field)
{

switch (field) {
case CSTRUCT_PORTAL_DAMAGE: return object_Integer(pNode->public_damage);
case CSTRUCT_PORTAL_FLOODING: return Tcl_NewDoubleObj(pNode->public_flooding);
case CSTRUCT_PORTAL_FROMID: return object_Integer(pNode->public_fromid);
case CSTRUCT_PORTAL_JUNCTION_TYPE: return object_Integer(pNode->public_junction_type);
case CSTRUCT_PORTAL_OPEN: return object_Integer(pNode->public_open);
case CSTRUCT_PORTAL_OPENING: return Tcl_NewDoubleObj(pNode->public_opening);
case CSTRUCT_PORTAL_RESERVED: return object_Boolean(pNode->public_reserved);
case CSTRUCT_PORTAL_TOID: return object_Integer(pNode->public_toid);
}

return NULL;
}

void Portal_AddLocation(Tcl_Interp *interp,Portal *p, Tcl_Obj *pValueDict) {
Tcl_Obj *tempval;

Tcl_DictObjPut(0,pValueDict,constant_stringObj(interp,"deckid"),Tcl_NewIntObj(p->deckid));

if(p->deckid) {
tempval=Tcl_NewObj();
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->deckid));
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->x));
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->y));
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->zoff));
Tcl_DictObjPut(interp,pValueDict,constant_stringObj(interp,"center"),tempval);
}
tempval=Tcl_NewObj();
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->nx));
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->ny));
Tcl_ListObjAppendElement(0, tempval, Tcl_NewIntObj(p->nz));
Tcl_DictObjPut(0,pValueDict,constant_stringObj(interp,"orientation"),tempval);
}

void Portal_SetType(Portal *p,SimType *pType) {
if(!p->pType) {
return;
}
Entity_SetType(p->pType,pType);
}

SimType *Portal_GetType(Portal *p) {
if(p->pType) {
return Entity_GetType(p->pType);
}
return NULL;
}

Tcl_Obj *Portal_ToDict(Tcl_Interp *interp,Portal *p) {
Tcl_Obj *pResult=NULL;
int i;
if(p->pType) {

```

```

Tcl_Obj *Portal_ToDict(Tcl_Interp *interp,Portal *p) {
    Tcl_Obj *pResult=NULL;
    int i;
    if(p->pType) {
        pResult=Entity_ToDict(interp,p->pType);
    }
    if(!pResult) {
        pResult=Tcl_NewObj();
    }
    /* Finally, Add the Tcl Data */
    for(i=0;i<CSTRUCT_PORTAL_Count;i++) {
        Tcl_Obj *newElement=Portal_StructGet(p,i);
        if(newElement) {
            Tcl_DictObjPut(interp,pResult,constant_stringObj(interp,Portal_paramNameMap[i].zName),newElement);
        }
    }
    return pResult;
}

```

```

/* Define the $StructName_StructChanged function */

```

```

int Portal_StructChanged(Portal *pNode,int field, int embargo, int force)
{
    if(field < 0) return 0;
    if(field > CSTRUCT_PORTAL_Count)
        return 0;
    if(!pNode->delta) force=1;
    /* No Photo, no comment */

    switch (field) {
        case CSTRUCT_PORTAL_DAMAGE:
            if(force) return 1;
            return pNode->public_damage != pNode->delta->public_damage;
        case CSTRUCT_PORTAL_FLOODING:
            if(force) {
                return pNode->public_flooding != 0.0;
            }
            return pNode->public_flooding != pNode->delta->public_flooding;

        case CSTRUCT_PORTAL_OPEN:
            if(force) return 1;
            return pNode->public_open != pNode->delta->public_open;
        case CSTRUCT_PORTAL_OPENING:
            if(force) {
                return pNode->public_opening != 0.0;
            }
            return pNode->public_opening != pNode->delta->public_opening;
    }
    return 0;
}

```

```

void Portal_StructAdvance(void) {
    Tcl_HashSearch search;
    Tcl_HashEntry *i;

    for(i=PortalFirst(&search); i ; i = Tcl_NextHashEntry(&search)) {
        Portal *p = (Portal*)Tcl_GetHashValue(i);
        Portal_UpdateGui(p);
        memcpy(p->delta,p,sizeof(*p));
    }
}

void Portal_Generic_Alloc(Portal *p)
{

    if (!p->delta) {
        p->delta=(Portal *) IRM_Alloc(sizeof(*p));
        memcpy(p->delta,p,sizeof(*p));
    }

}

void Portal_Generic_Free(Portal *p)
{

    if (p->delta) {
        IRM_Free((char *)p->delta);
        p->delta=0;
    }

}

Tcl_Obj *Portal_Identify(Portal *pNode) {
    if(!pNode) {
        return Tcl_NewWideIntObj(0);
    }
    return Tcl_NewWideIntObj(pNode->id);
}
Tcl_Obj *Portal_TypeId(Portal *pNode) {
    SimType *pType;
    if(!pNode) {
        return Tcl_NewIntObj(0);
    }
    pType=Portal_GetType(pNode);
}

```

```

simType *pType;
if(!pNode) {
    return Tcl_NewIntObj(0);
}
pType=Portal_GetType(pNode);
if(!pType) {
    return Tcl_NewIntObj(0);
}
return Tcl_NewWideIntObj(pType->id);
}

/*
** Find a Portal given a Tcl_Obj that contains the Portal ID.
** Leave an error message in interp and return TCL_ERROR if anything
** goes wrong.
*/
int Portal_FromTclObj(Tcl_Interp *interp, Tcl_Obj *pObj, Portal **ppPortal){
    Roid i;

    if( Tcl_GetIntFromObj(interp, pObj, &i) ) {
        Tcl_Obj *substring=qdStringObjLeftTrim(pObj,'p');
        if (!substring) {
            return TCL_ERROR;
        }
        if(Tcl_GetIntFromObj(interp, substring, &i)) {
            Tcl_DecrRefCount(substring);
            return TCL_ERROR;
        }
        Tcl_DecrRefCount(substring);
        Tcl_ResetResult(interp);
    }
    *ppPortal = PortalById(i, 0);
    if( *ppPortal==0 ){
        Tcl_AppendResult(interp, "no such portal: ",
            Tcl_GetStringFromObj(pObj, 0), 0);
        return TCL_ERROR;
    }
    return TCL_OK;
}

/*
** Given the name of one of the arState[] values in the Portal_ structure,
** return the index of the particular arState[]. Return TCL_OK.
**
** Leave an error message in interp and return TCL_ERROR if
** anything goes wrong.
*/

```

```

/*
** Given the name of one of the arState[] values in the Portal_ structure,
** return the index of the particular arState[]. Return TCL_OK.
**
** Leave an error message in interp and return TCL_ERROR if
** anything goes wrong.
*/
int Portal_ValueOffset(
    Tcl_Interp *interp,
    Tcl_Obj *pObj,
    int *pIndex,
    int *pType
){
    int lo, hi, mid, c, max, i;
    int nName;
    const char *zName;
    const struct IrmParamNameMap *aParam = Portal_paramNameMap;

    lo = 0;
    hi = max = CSTRUCT_PORTAL_Count - 1;
    zName = Tcl_GetStringFromObj(pObj, &nName);
    mid = (lo+hi)/2;

    while( lo<=hi ){
        mid = (lo+hi)/2;
        c = strcmp(zName, aParam[mid].zName, nName);
        if( c<0 ){
            hi = mid-1;
        }else if( c>0 ){
            lo = mid+1;
        }else if(
            (mid>0 && strcmp(zName, aParam[mid-1].zName, nName)==0) ||
            (mid<max && strcmp(zName, aParam[mid+1].zName, nName)==0)
        ){
            i = mid;
            while( i>0 && strcmp(zName, aParam[i-1].zName, nName)==0 ){
                i--;
            }
            if( strlen(aParam[i].zName)==nName ){
                *pIndex = aParam[i].iCode;
                *pType = aParam[i].pType;
                return TCL_OK;
            }
            if(interp) {
                Tcl_AppendResult(interp, "ambiguous parameter:", 0);
                do{
                    Tcl_AppendResult(interp, " ", aParam[i++].zName, 0);
                }while( i<=max && strcmp(zName, aParam[i].zName, nName)==0 );
            }
            return TCL_ERROR;
        }else{
            *pIndex = aParam[mid].iCode;
            *pType = aParam[mid].pType;
            return TCL_OK;
        }
    }
    if(interp) {
        Tcl_AppendResult(interp, "unknown parameter \"", zName,
            "\" - nearby choices:", 0);
        for(i=mid-3; i<mid+3; i++){
            if( i<0 || i>max ) continue;
            Tcl_AppendResult(interp, " ", aParam[i].zName, 0);
        }
    }
    return TCL_ERROR;
}

/*
** title:

```

```

}
/*
** title:
*/
int Portal_nodereval(
    Tcl_Interp *interp,
    Portal *p,
    Tcl_Obj *body
) {
    Tcl_Obj *pValueDict;
    int i;
    Tcl_Obj **varv;
    int varc,result;

    pValueDict=Portal_ToDict(interp,p);
    if (Tcl_ListObjGetElements(interp, pValueDict, &varc, &varv) != TCL_OK) {
        return TCL_ERROR;
    }
    for(i=0;i<varc;i+=2) {
        Tcl_ObjSetVar2(interp, varv[i], (Tcl_Obj *)NULL, varv[i+1], 0);
    }
    Tcl_ObjSetVar2(interp,constant_stringObj(interp,"id"),NULL,Portal_Identify(p),0);
    Tcl_ObjSetVar2(interp,constant_stringObj(interp,"typeid"),NULL,Portal_TypeId(p),0);

    result=Tcl_EvalObjEx(interp, body, 0);
    if(result!=TCL_OK) {
        if(pValueDict) {
            Tcl_DecrRefCount(pValueDict);
        }
        return result;
    }
    {
        /*
        ** Read values back into the dict
        ** For now, we limit writeback to state variables
        ** And we don't care about garbage values
        */
        for(i=0;i<varc;i+=2) {
            Tcl_Obj *newValue;
            int offset;
            int type;
            newValue=Tcl_ObjGetVar2(interp,varv[i],(Tcl_Obj *)NULL,0);
            if(newValue==varv[i+1]) {
                /* Undocumented, unsanctioned, but it works in practice
                ** If the pointer hasn't changed, neither has the value
                */
                continue;
            }
            if(!newValue) {
                /* Variable must have been unset... move along */
                continue;
            }
            if( Portal_ValueOffset(0, varv[i], &offset, &type) == TCL_OK ) {
                Portal_StructSet(interp,p,offset,newValue);
            } else {
                Portal_DictPut(p,varv[i],newValue);
            }
        }
        Portal_ApplySettings(p);
    }
    if(pValueDict) {
        Tcl_DecrRefCount(pValueDict);
    }
    return TCL_OK;
}

```

# Code Generation

The implementation of the Portal Structure example was 455 lines.  
(Including comments.)

# The Product

portal.tcl

# The Product

portal.tcl

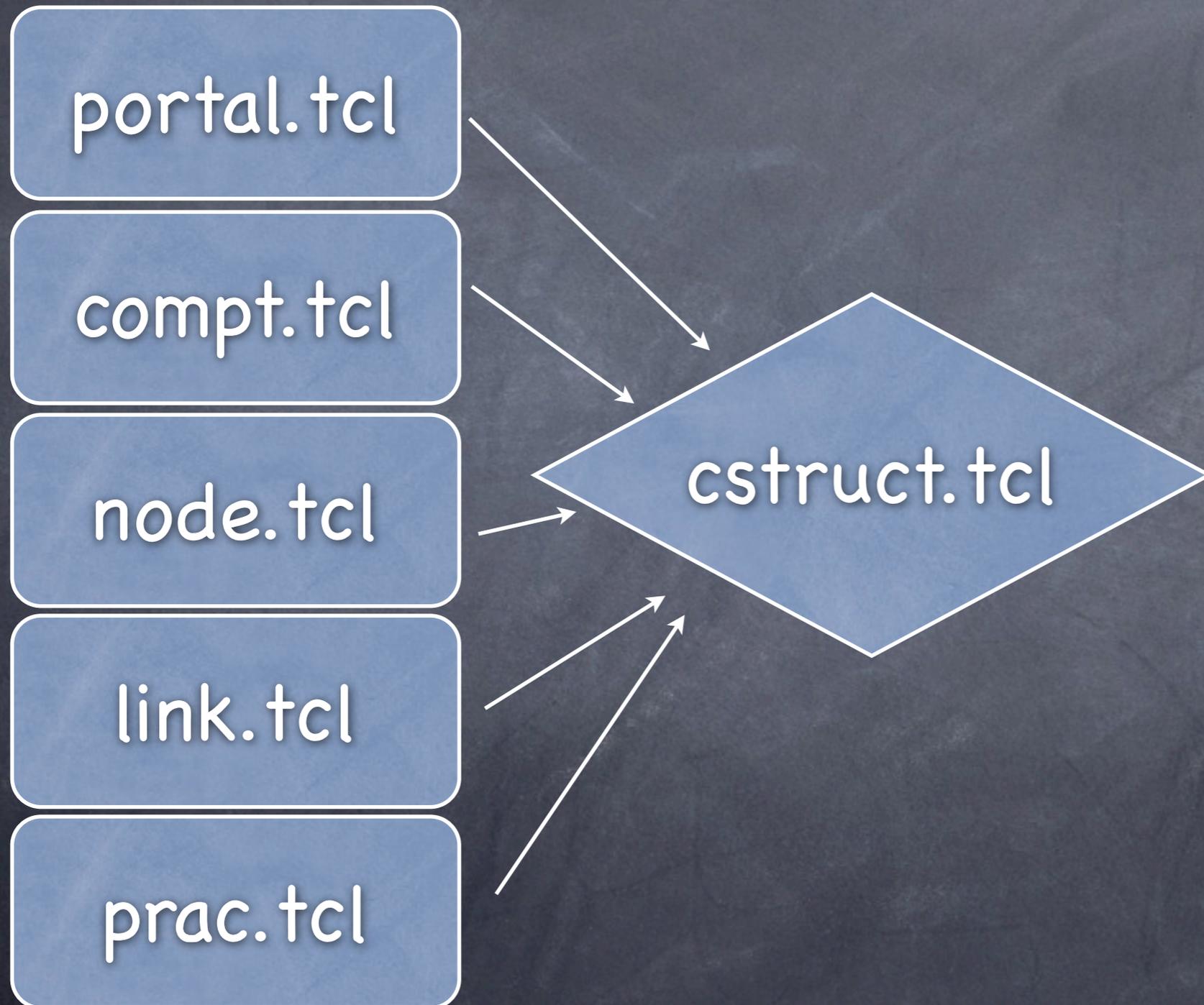
compt.tcl

node.tcl

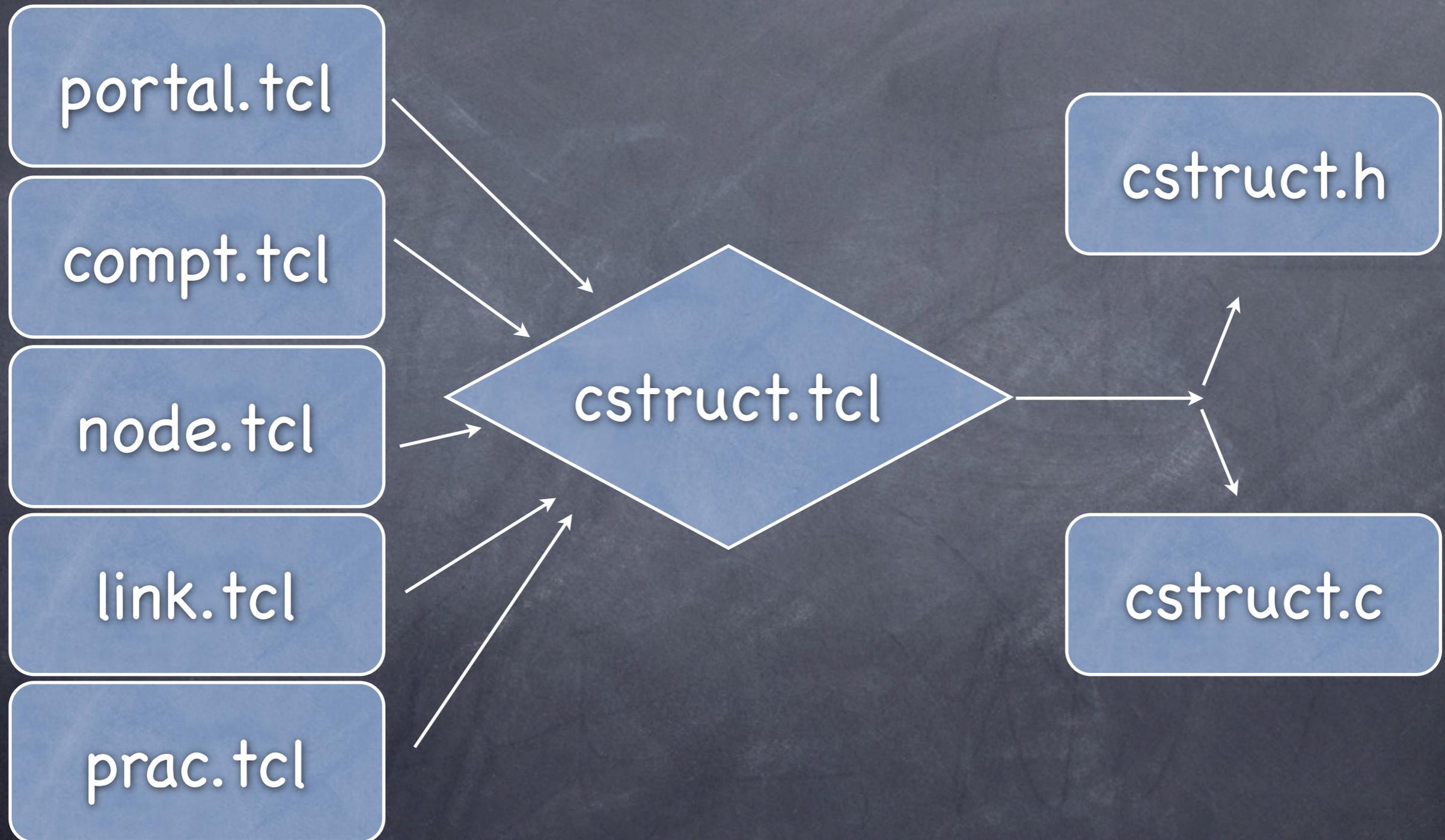
link.tcl

prac.tcl

# The Product



# The Product

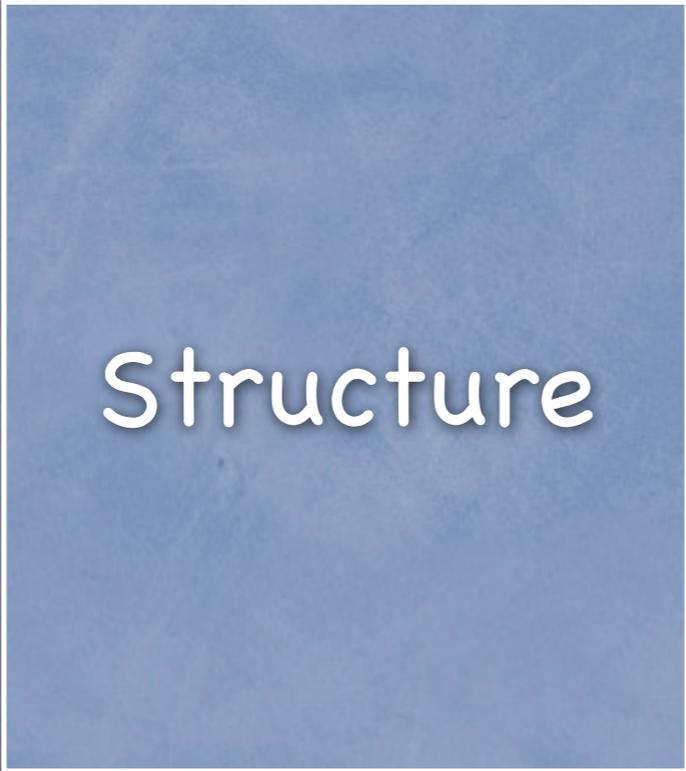


KEEP YOUR FORK



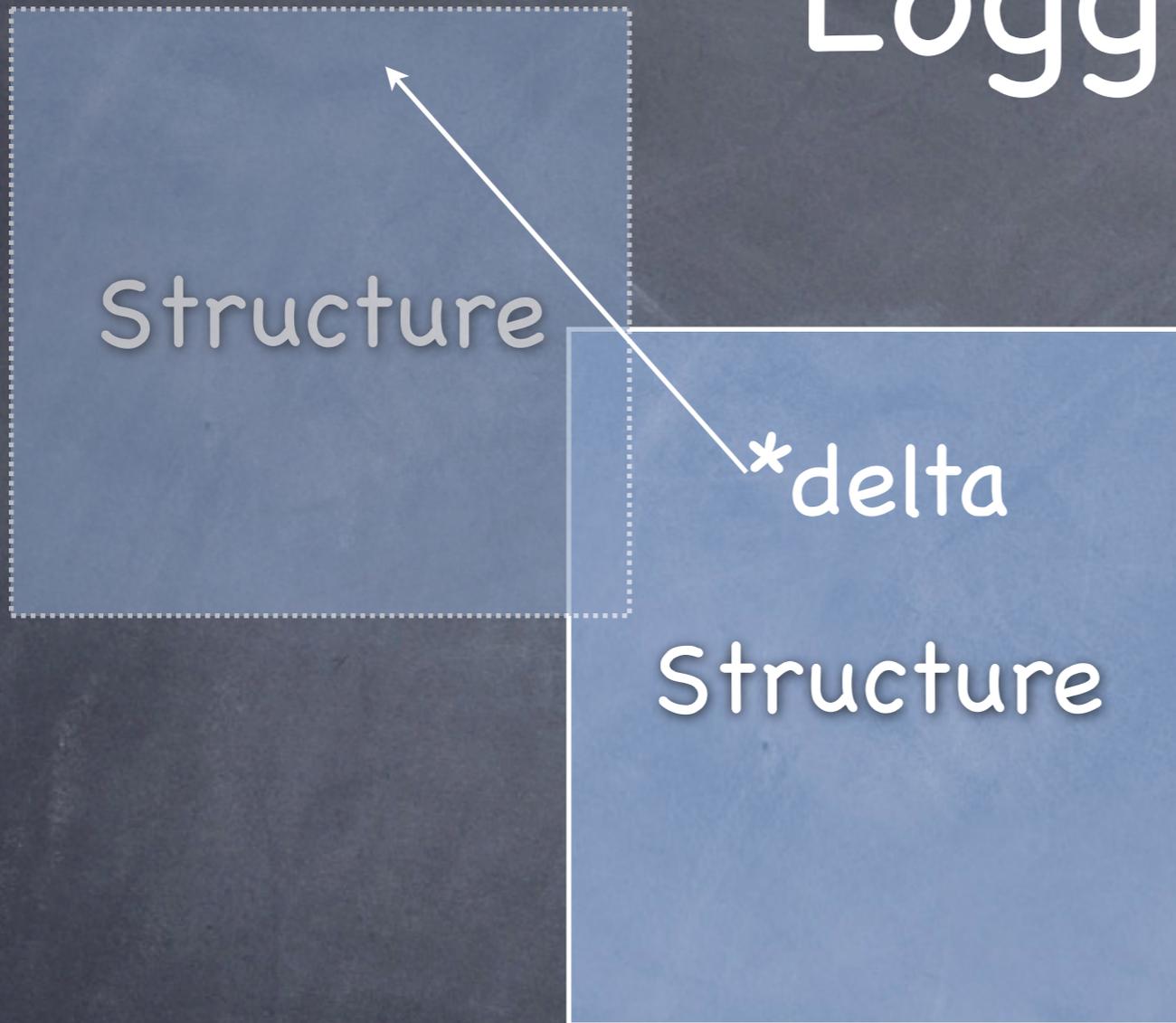
*There's Pie!*

# Logging

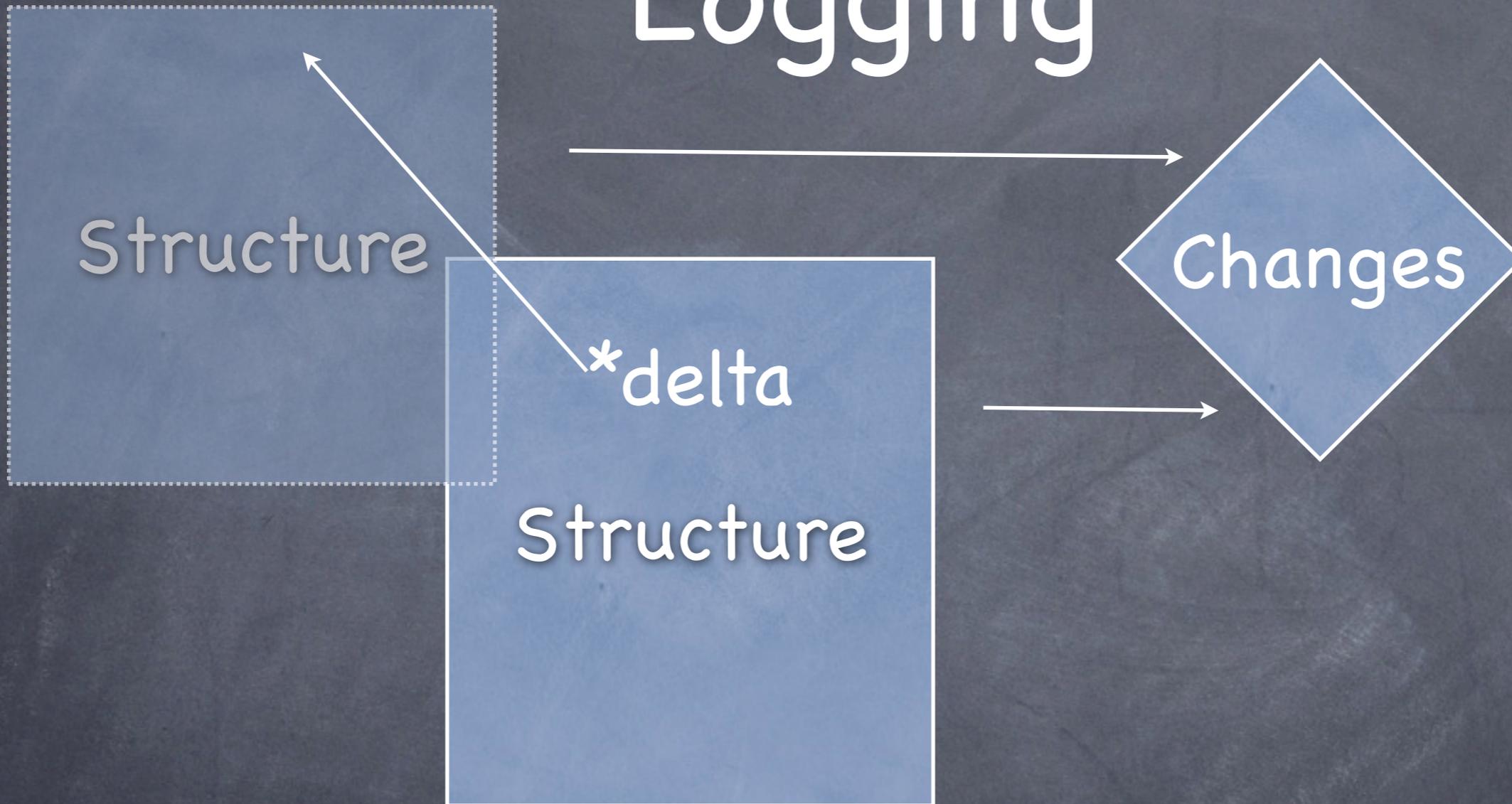


Structure

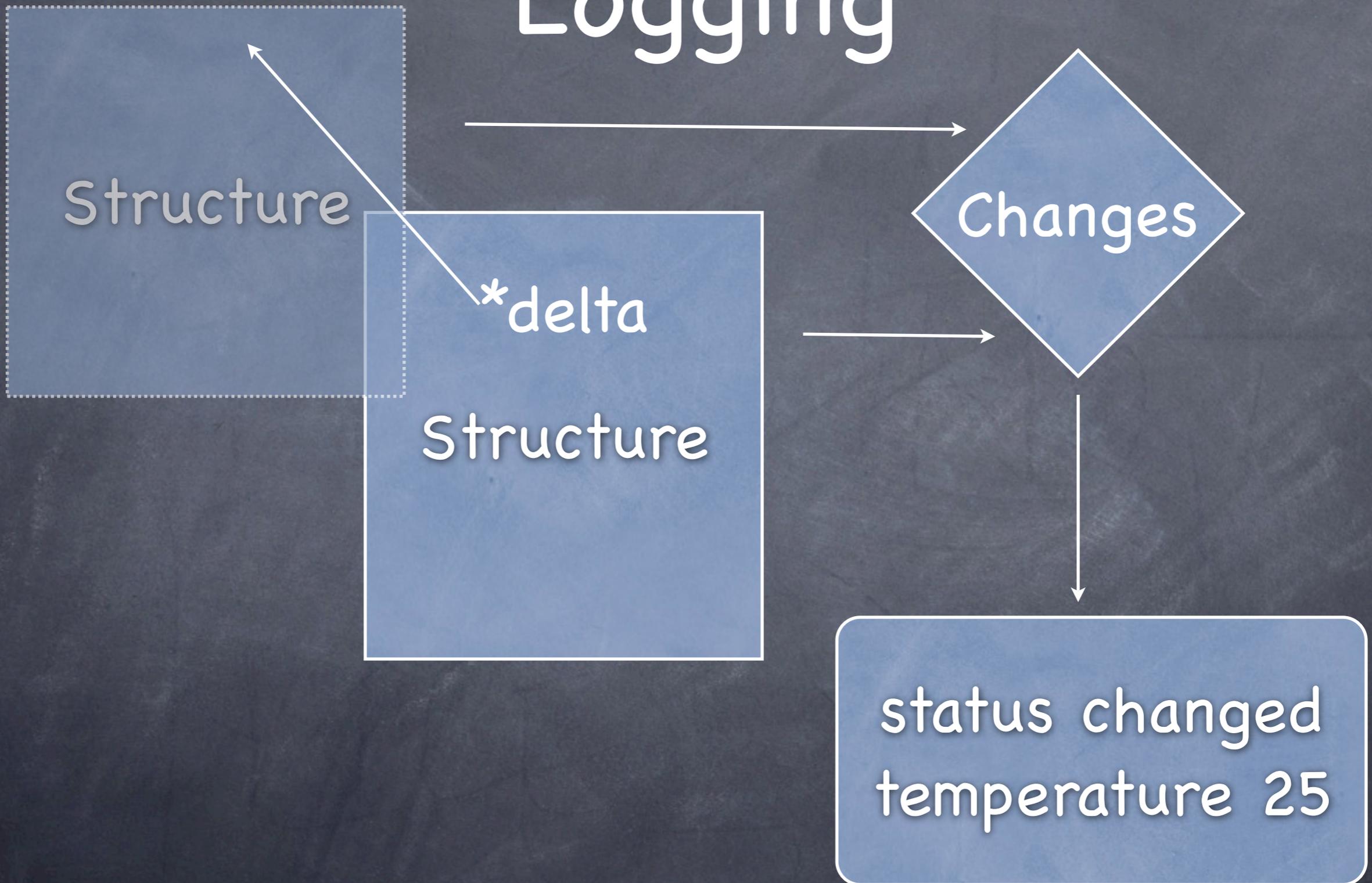
# Logging



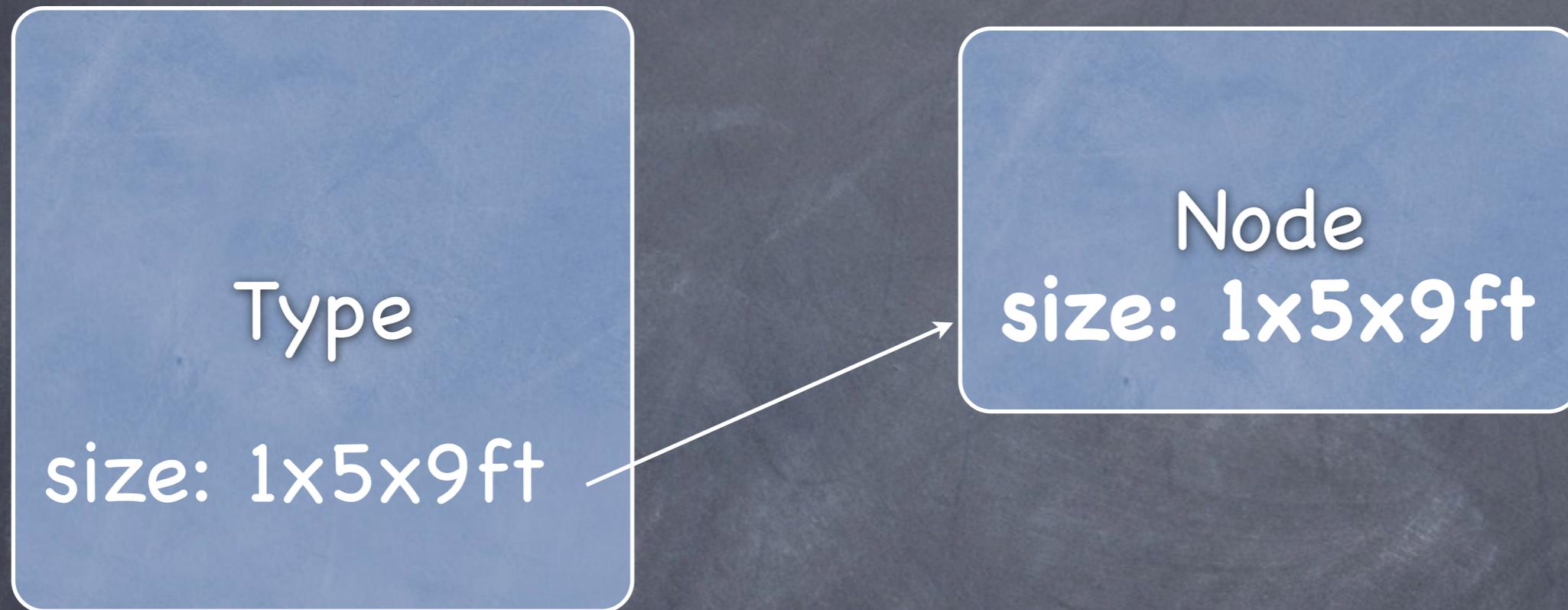
# Logging



# Logging



# Property Inheritance



Demo