PROTOCOL VERIFICATION IN THE COMPUTATIONAL MODEL

(using EasyCrypt)

NAXOS

$\mathcal{A}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{B}$

$esk_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{\lambda}$ $\qquad$ $X = g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}$ $\longrightarrow$

$esk_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{\lambda}$

$\longleftarrow$ $Y = g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}$

$$\mathcal{B}: \; K \leftarrow H_2(pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \; X^{sk_{\mathcal{B}}}, \; X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \; \mathcal{A}, \; \mathcal{B})$$

$$\mathcal{A}: \; K \leftarrow H_2(Y^{sk_{\mathcal{A}}}, \; pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, \; Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, \; \mathcal{A}, \; \mathcal{B})$$

AKE PROTOCOLS

$\mathcal{A}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{B}$

$esk_{\mathcal{A}} \xleftarrow{\$} \{0,1\}^{\lambda}$ $\qquad$ $X = g^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}$ $\longrightarrow$

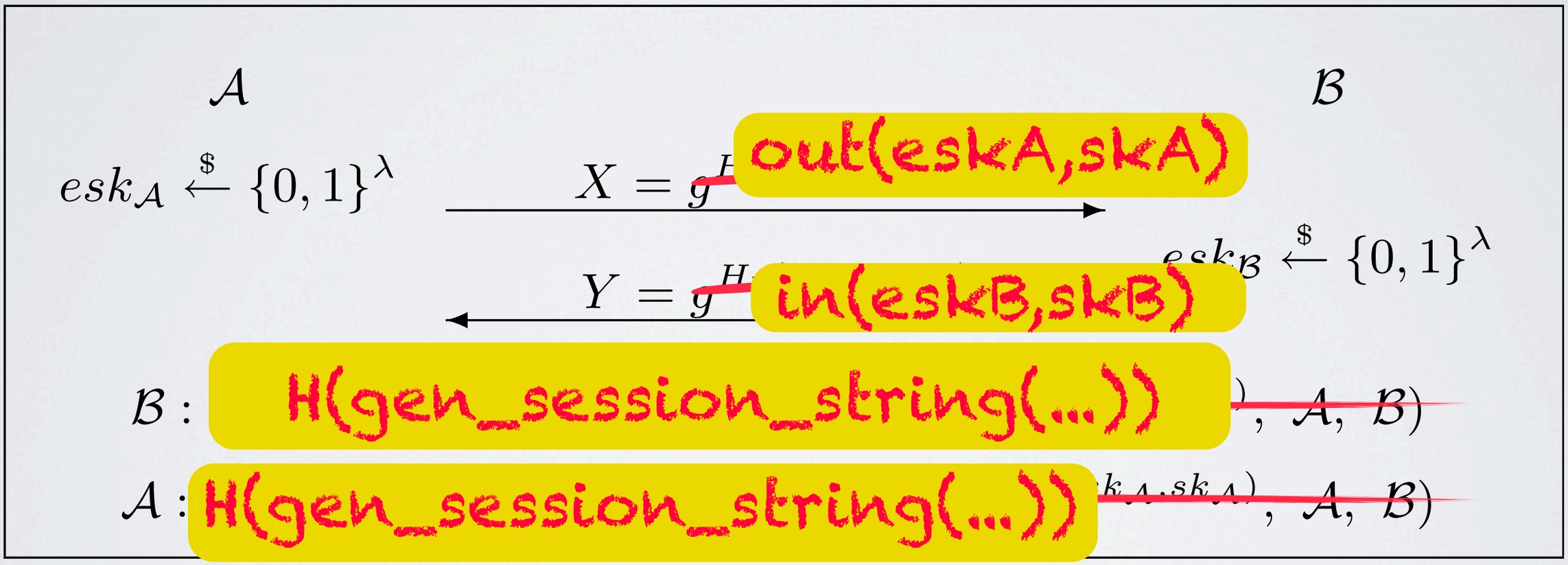
$esk_{\mathcal{B}} \xleftarrow{\$} \{0,1\}^{\lambda}$

$\longleftarrow$ $Y = g^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}$

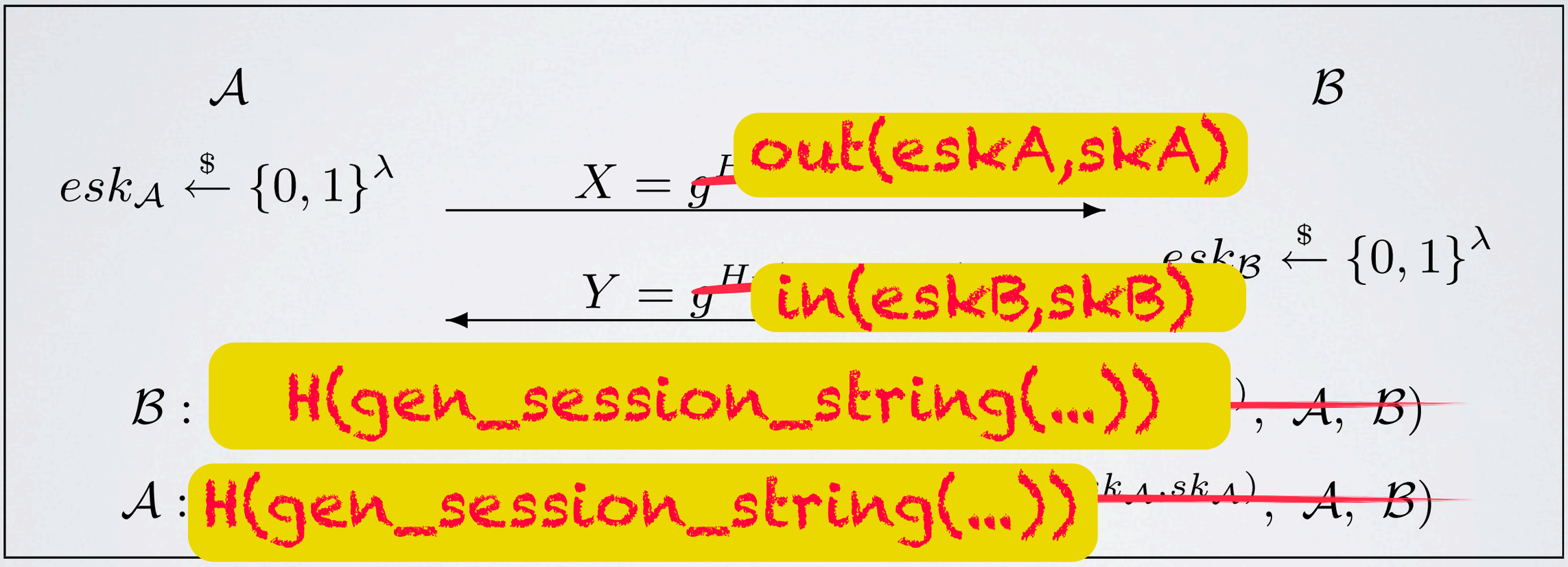
$\mathcal{B}: K \leftarrow H_2(pk_{\mathcal{A}}^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, X^{sk_{\mathcal{B}}}, X^{H_1(esk_{\mathcal{B}}, sk_{\mathcal{B}})}, \mathcal{A}, \mathcal{B})$

$\mathcal{A}: K \leftarrow H_2(Y^{sk_{\mathcal{A}}}, pk_{\mathcal{B}}^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, Y^{H_1(esk_{\mathcal{A}}, sk_{\mathcal{A}})}, \mathcal{A}, \mathcal{B})$

AKE PROTOCOLS



AKE PROTOCOLS



how much of the proof can we get away with without instantiating?

ORACLES

- KG() : public_key
- Init(A : part, B: part) : msg option
- Respond(B : part, A : part, X : msg) : msg option
- Complete(A:part, B:part, X:msg, Y:msg) : unit
- Hash(str : session_string) : session_key
- Corrupt(A : part) : private_key option
- EphKeyReveal(A : part, X : msg) : eph_key option
- KeyReveal(s : session_id) : session_key
- Test(s : session_id) : session_key

ORACLES (HASH, 4 LOC)

```
fun H(lam:session_string) : session_key = {
 var h : session_key = gen_session_key(0);
 if (!in_dom(lam, LH)) { LH[lam] = h; }
 return LH[lam];
}
```

ORACLES (INIT, ~10 LOC)

```
fun Init(A:part, B:part) : message option = {
  var x : eph_key = gen_eph_key(0);
  var a : secret_key;
  var X : message option = None;
  if (in_dom(A, skey) && in_dom(B, skey)){
   a = skey[A];
   X = Some(out_noclash(a, x));
   if (!in_dom((proj(X),A), seed)) {
    incomplete_sessions[(A,proj(X))] = (B,false);
    seed[(proj(X),A)] = x;
   }
   else
   {
    X = None;
   }
  }
  return X;}
```

ORACLES (KEYREVEAL, ~100 LOC)

```
fun KeyReveal(s : session_id) : session_key = {
  var A:part = fstpart(s);
  var B:part = sndpart(s);
  var X:message = fstmsg(s);
  var Y:message = sndmsg(s);
  var x : eph_key = seed[(X,A)];
  var B' : part = dummy_part;
  var Y' : message = dummy_msg;
  var eph_flagA : bool = false;
  var eph_flagB : bool = false;
  var A' : part = dummy_part;
  var X' : message = dummy_msg;
  var sstr : session_string = dummy_string;
  var ssskey : session_key = dummy_session_key;
  var matchb : bool = false;
  var h : session_key = dummy_session_key;
  var sidA, sidB : session_id;
  h = gen_session_key(0);
  if (in_dom((A,X), complete_sessions))
  { (* (A,_,X,_) is completed*)
    B' = session_part(complete_sessions[(A,X)]);
    Y' = session_msg(complete_sessions[(A,X)]);
    sidA = mk_sid(A, B', X, Y');
    eph_flagA = session_eph_flag(complete_sessions[(A,X)]);
    if ( B = B' && Y = Y')
    {(* B = B' /\ Y = Y'*)
    if (!in_dom((B,Y), complete_sessions))
     {(*B,_,Y,_ not complete*)
      if (! in_dom(sidA, tested_session))
      {(*Fresh*)
        complete_sessions[(A,X)] = mk_session_descr(B',Y',eph_flagA,true);
        sstr = gen_session_string_sid(sidA,skey,seed);
            (*sstr = gen_session_string(skey[A], x, B,Y);*)
            ssskey = iH(sstr, h);
      }
      else
      { (*not fresh*)
        ssskey = dummy_session_key;
      }
    }
  else
  { (* B,_,Y_ complete *)
      A' = session_part(complete_sessions[(B,Y)]);
      X' = session_msg(complete_sessions[(B,Y)]);
      sidB = mk_sid(B, A', Y, X');
      matchb = session_match( mk_sid (A, B', X,Y'),mk_sid(B,A',Y,X'));
```

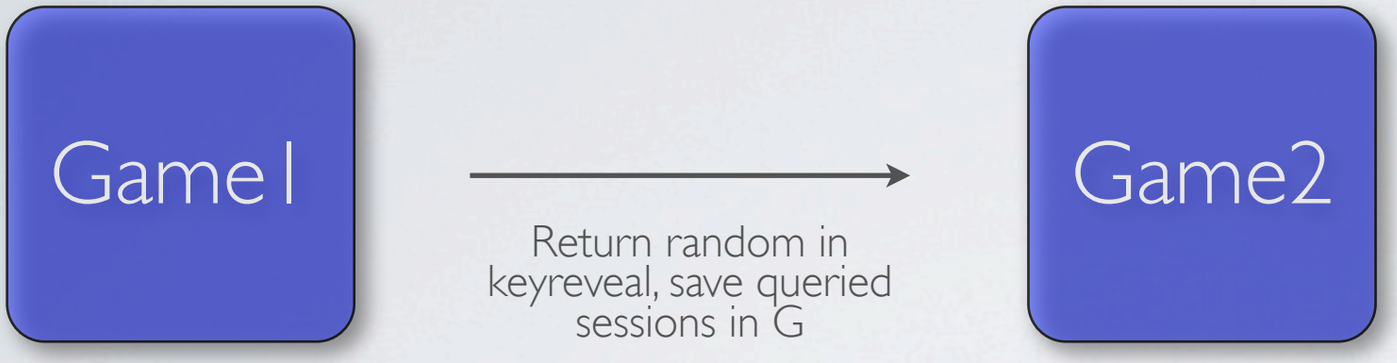
MAIN

```
fun Main () : bool = {
 var b' : bool;
 var tt : unit;
 complete_sessions = empty_map;  incomplete_sessions = empty_map;
 corrupt = empty_map;  pkey = empty_map;
 skey = empty_map;  LH  = empty_map;
 LHT  = empty_map;  seed  = empty_map;
 tested_session = empty_map;  G = empty_map;
 b = {0,1};
 b' = A();
 return (b = b');
}
```

GAME SEQUENCE

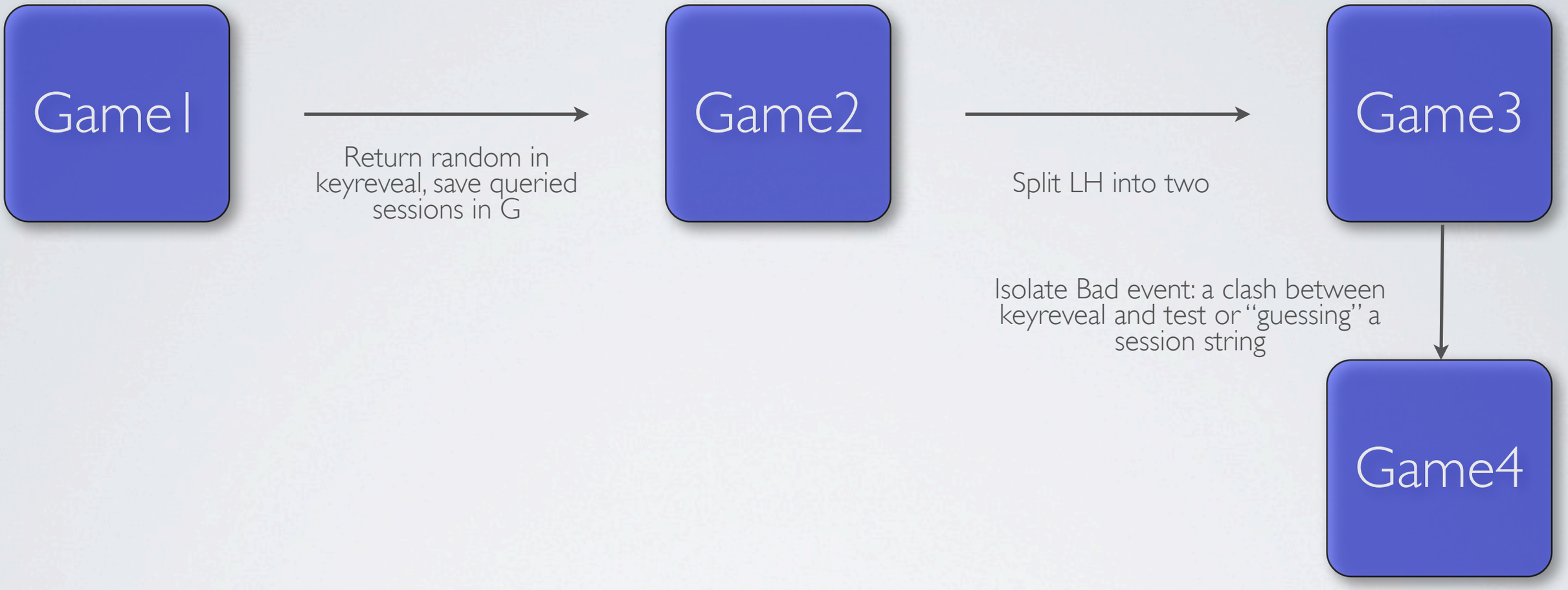
Game I

GAME SEQUENCE

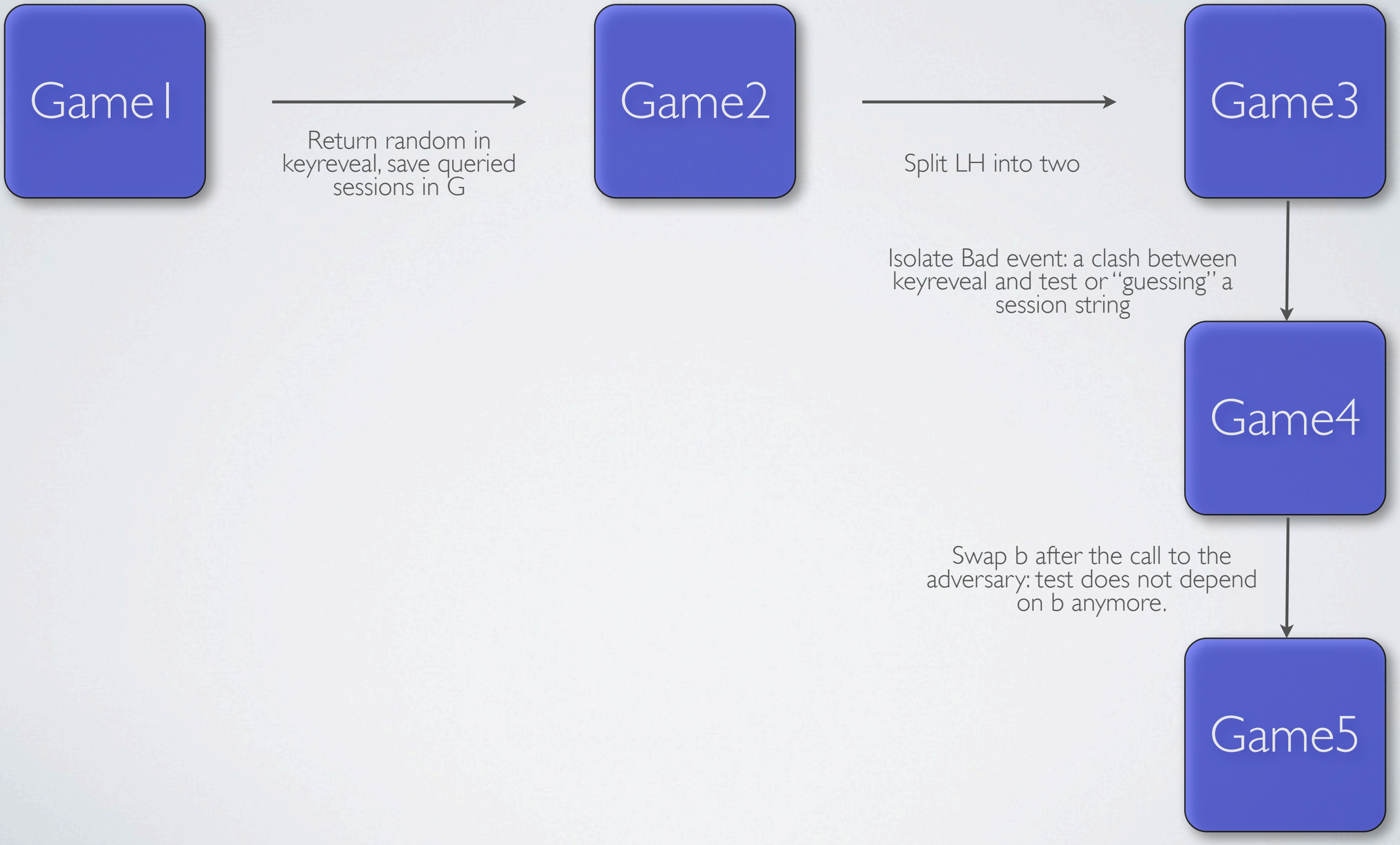


GAME SEQUENCE

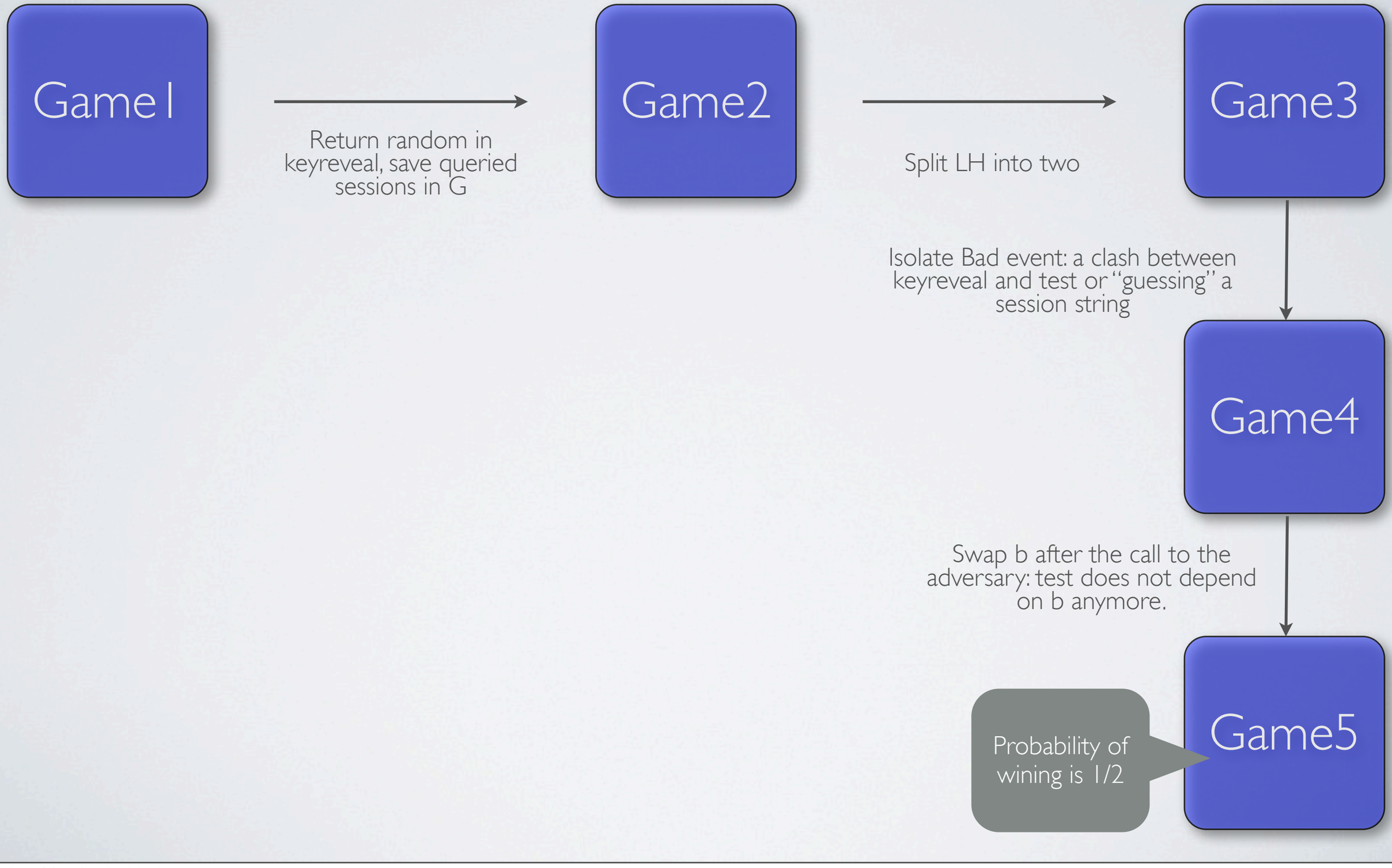
GAME SEQUENCE



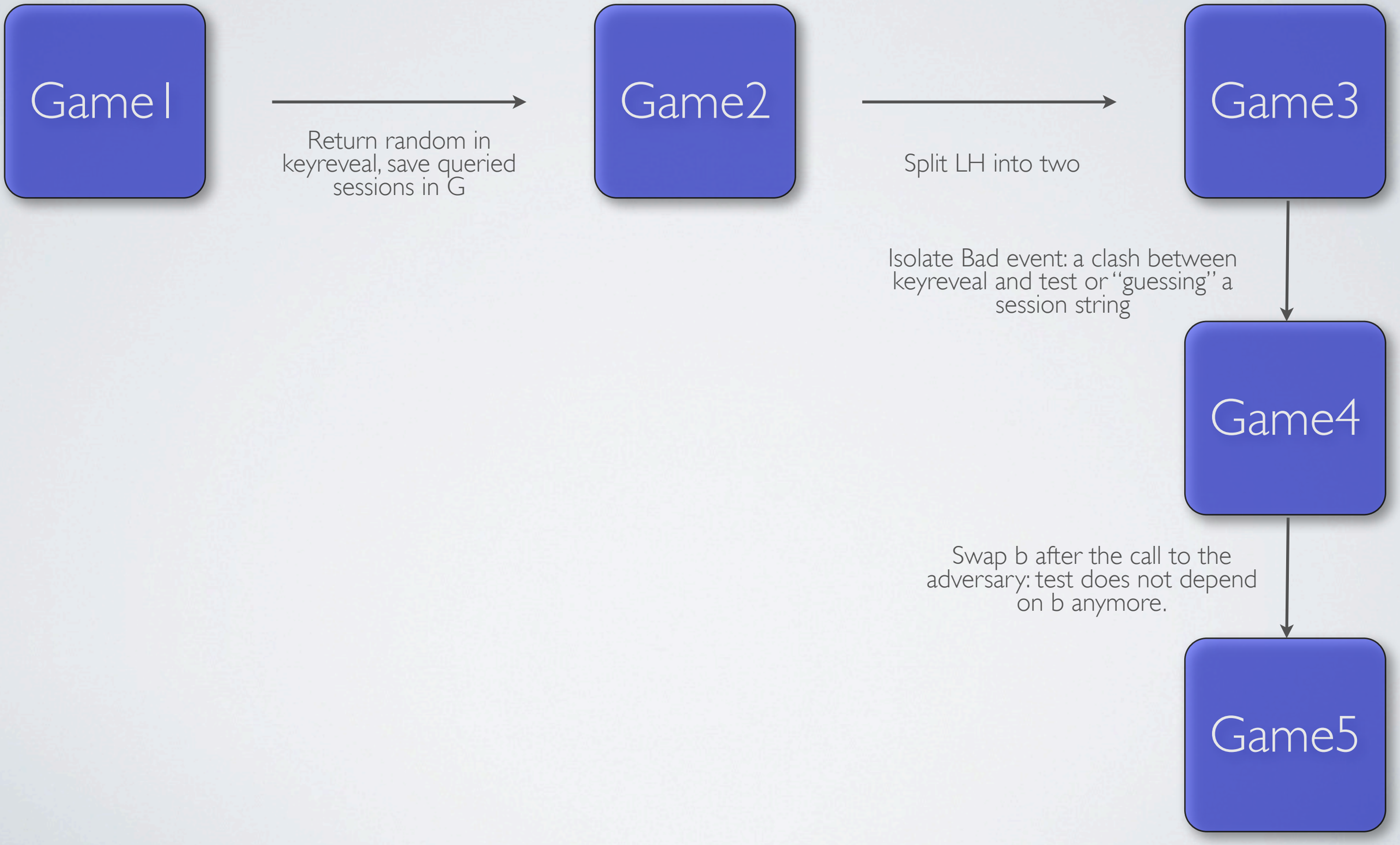
GAME SEQUENCE



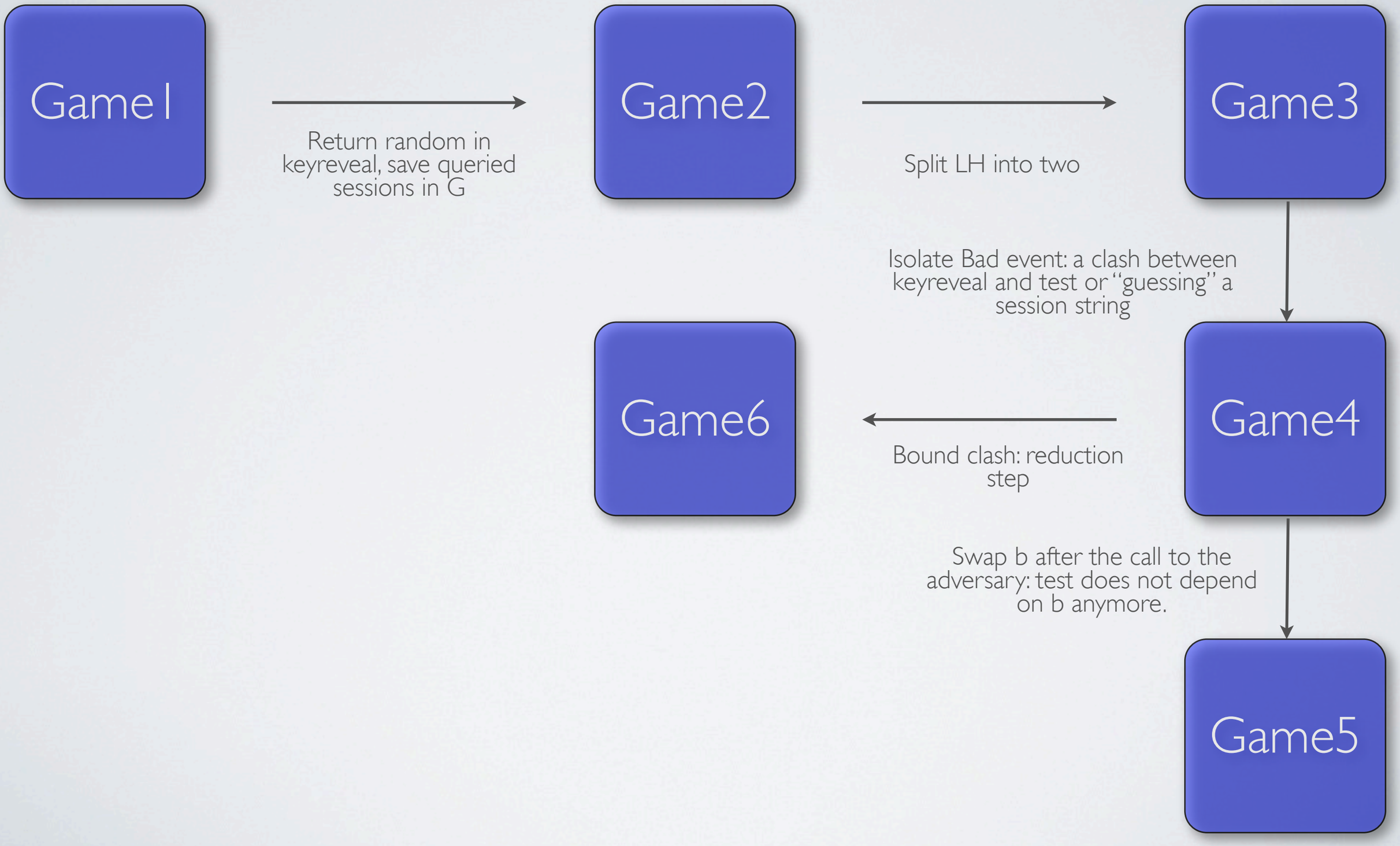
GAME SEQUENCE



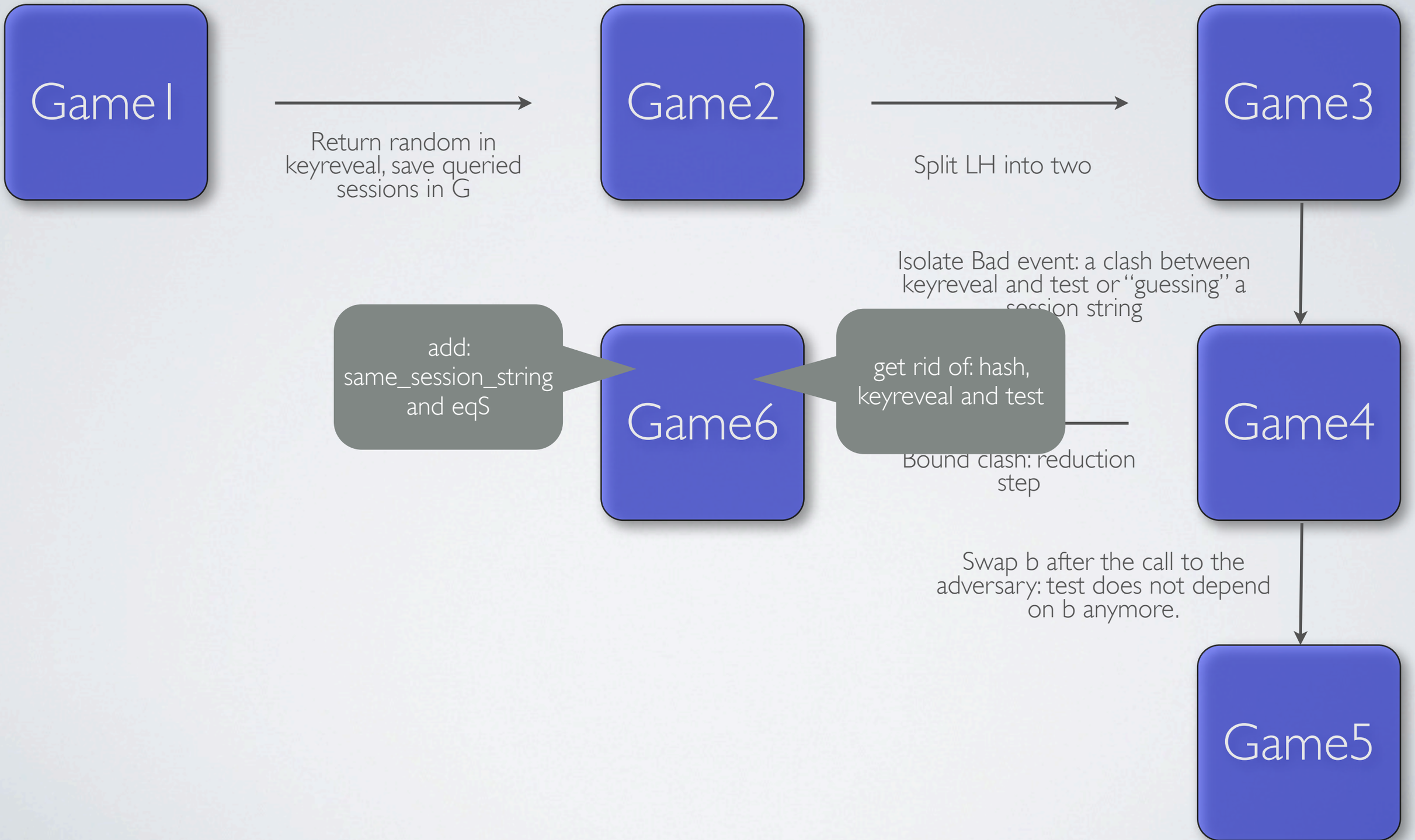
GAME SEQUENCE



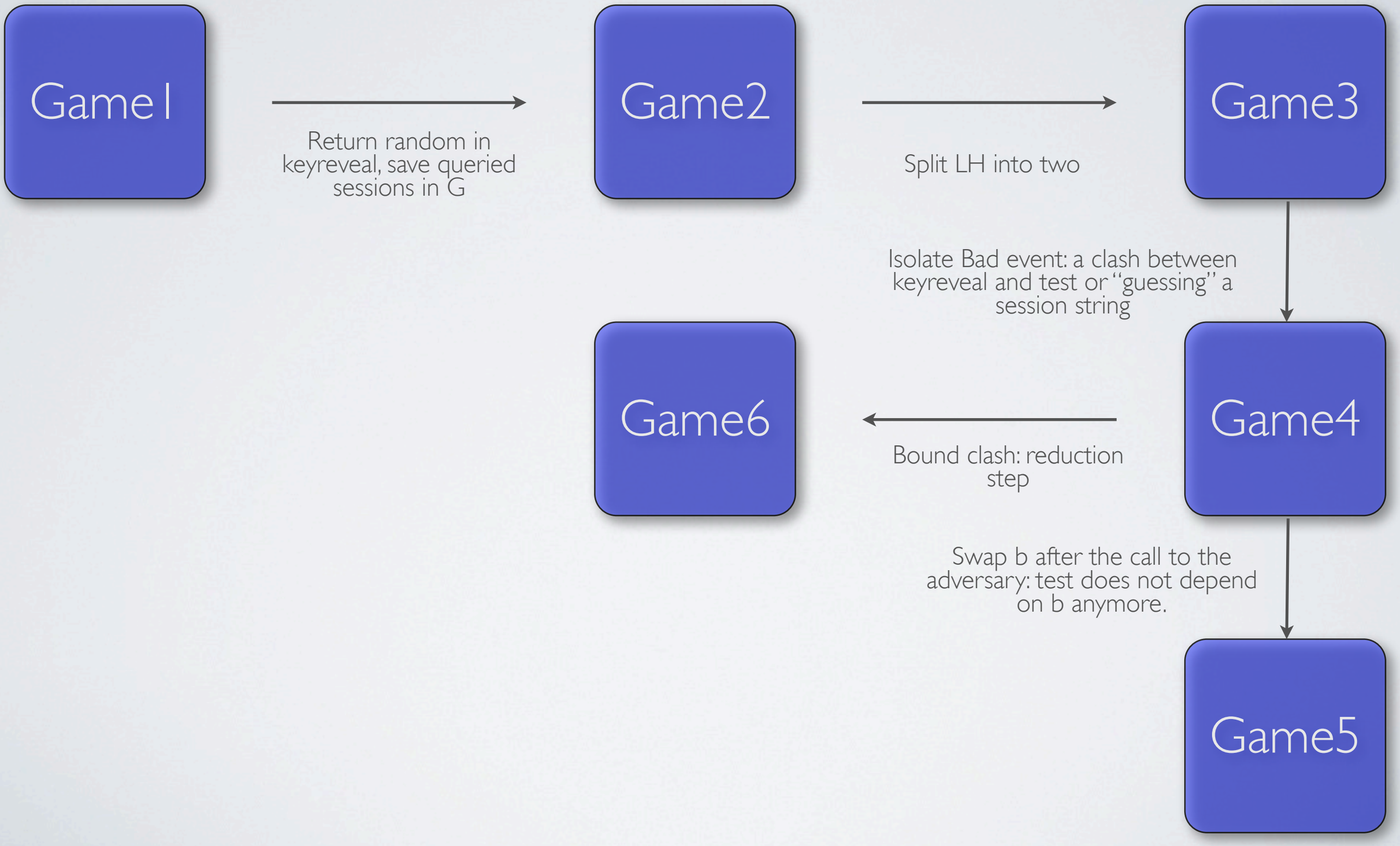
GAME SEQUENCE



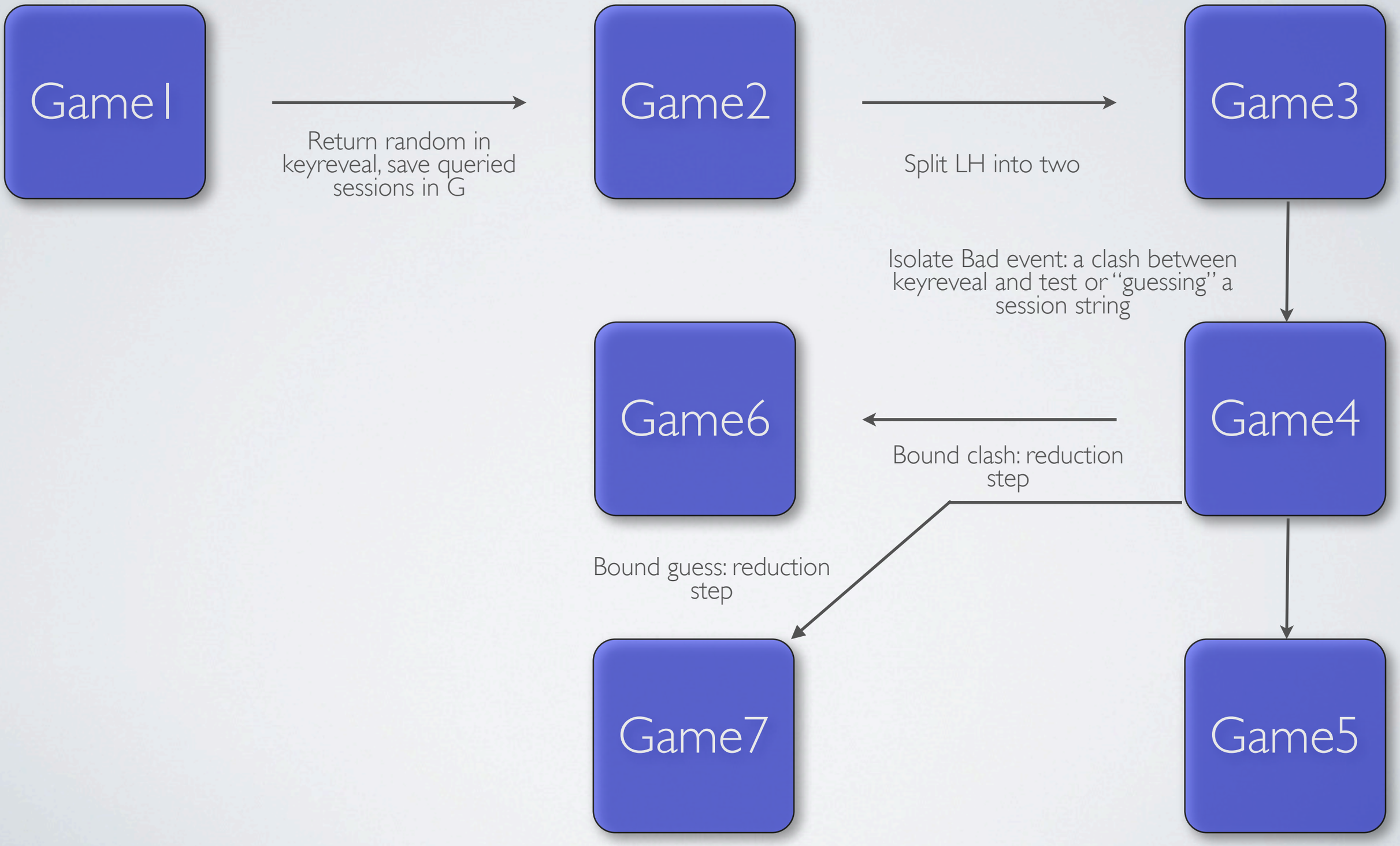
GAME SEQUENCE



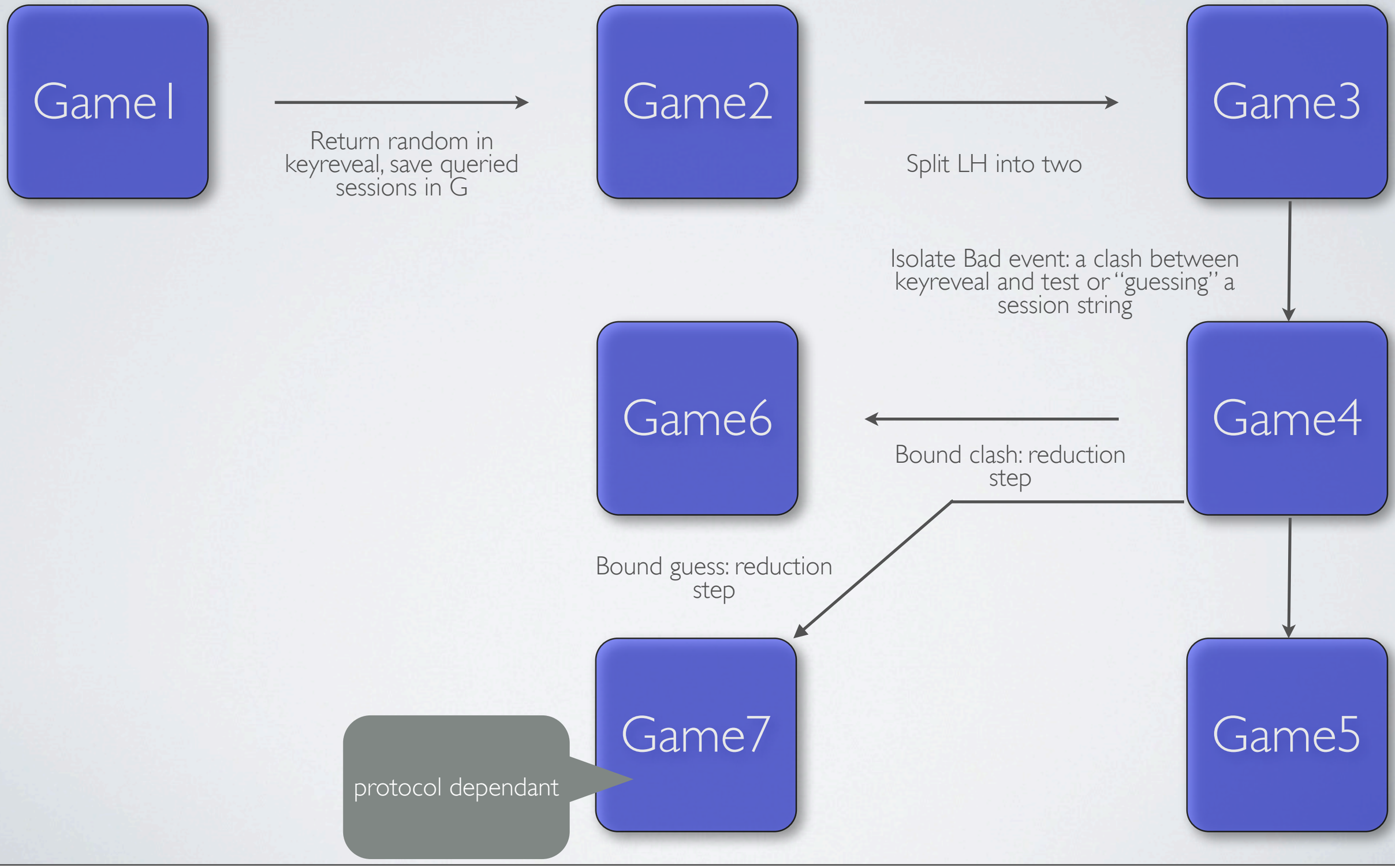
GAME SEQUENCE



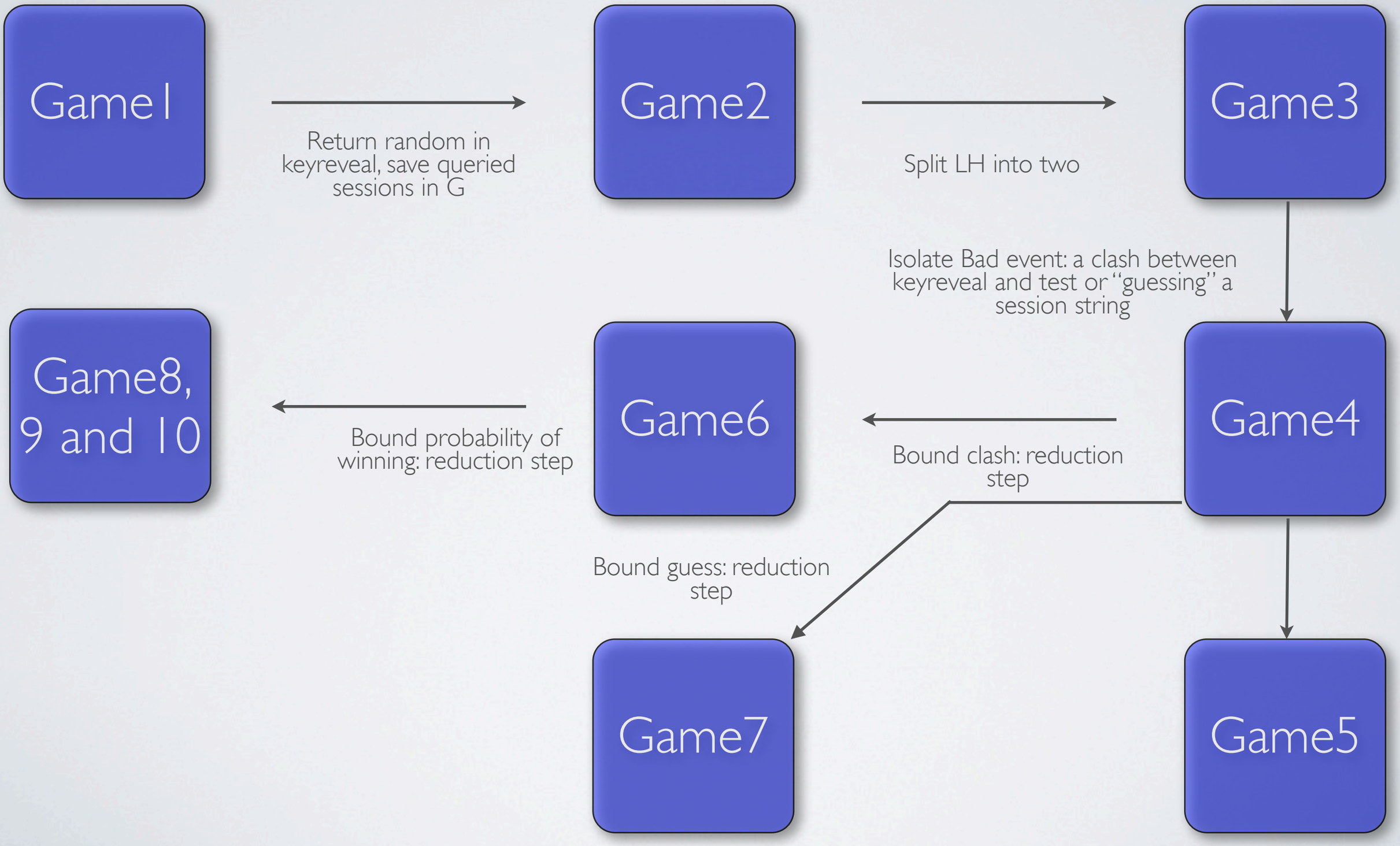
GAME SEQUENCE



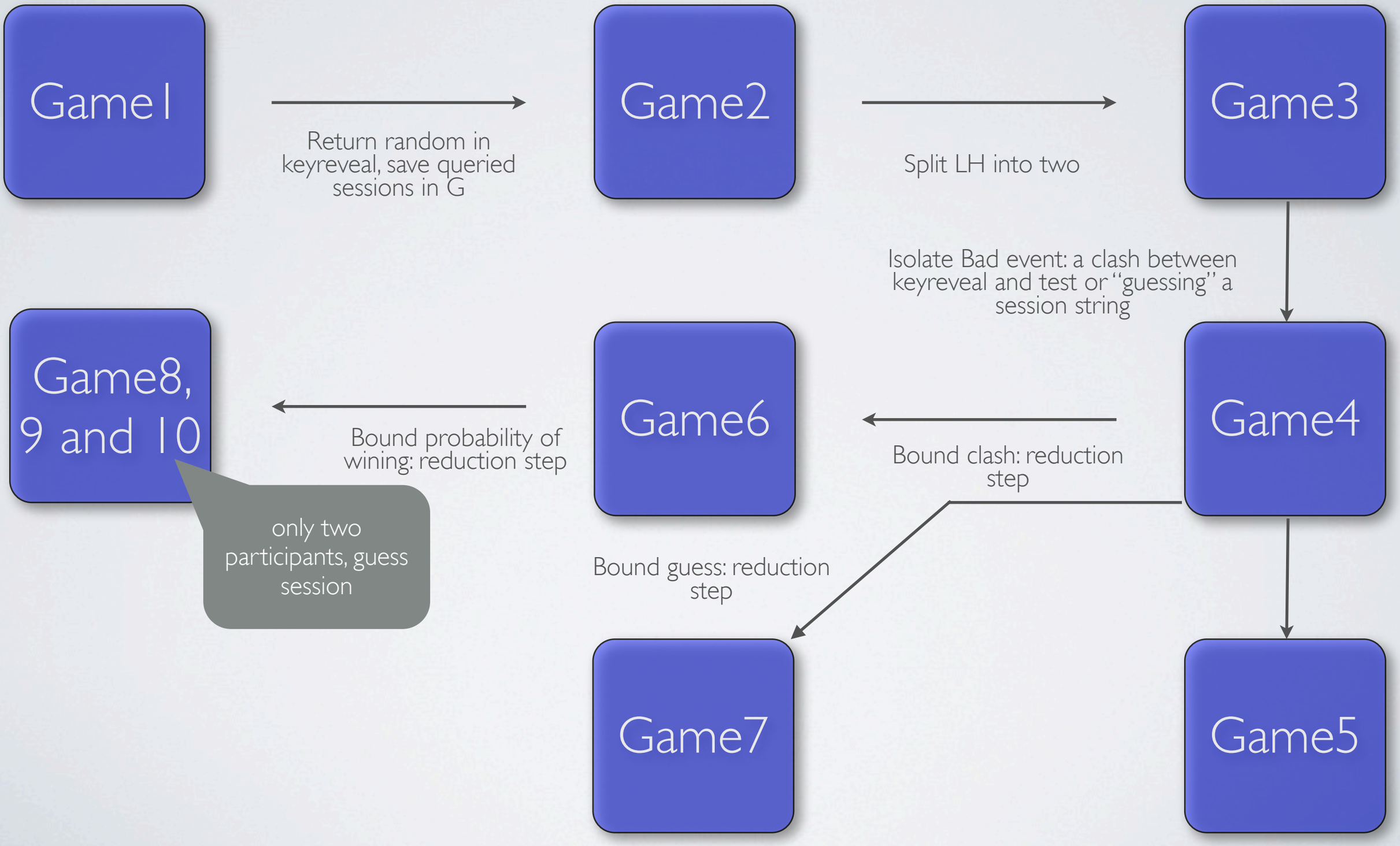
GAME SEQUENCE



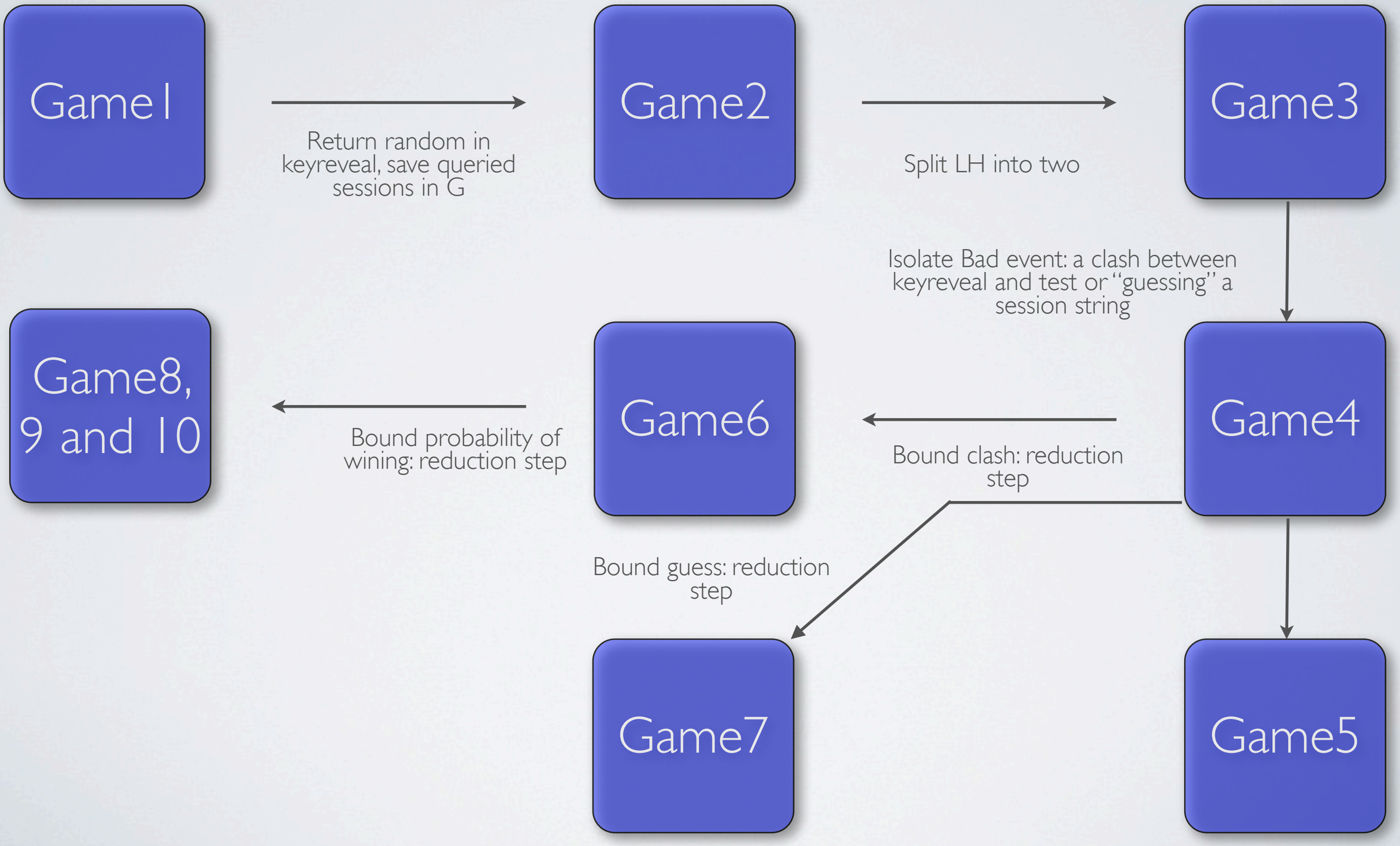
GAME SEQUENCE



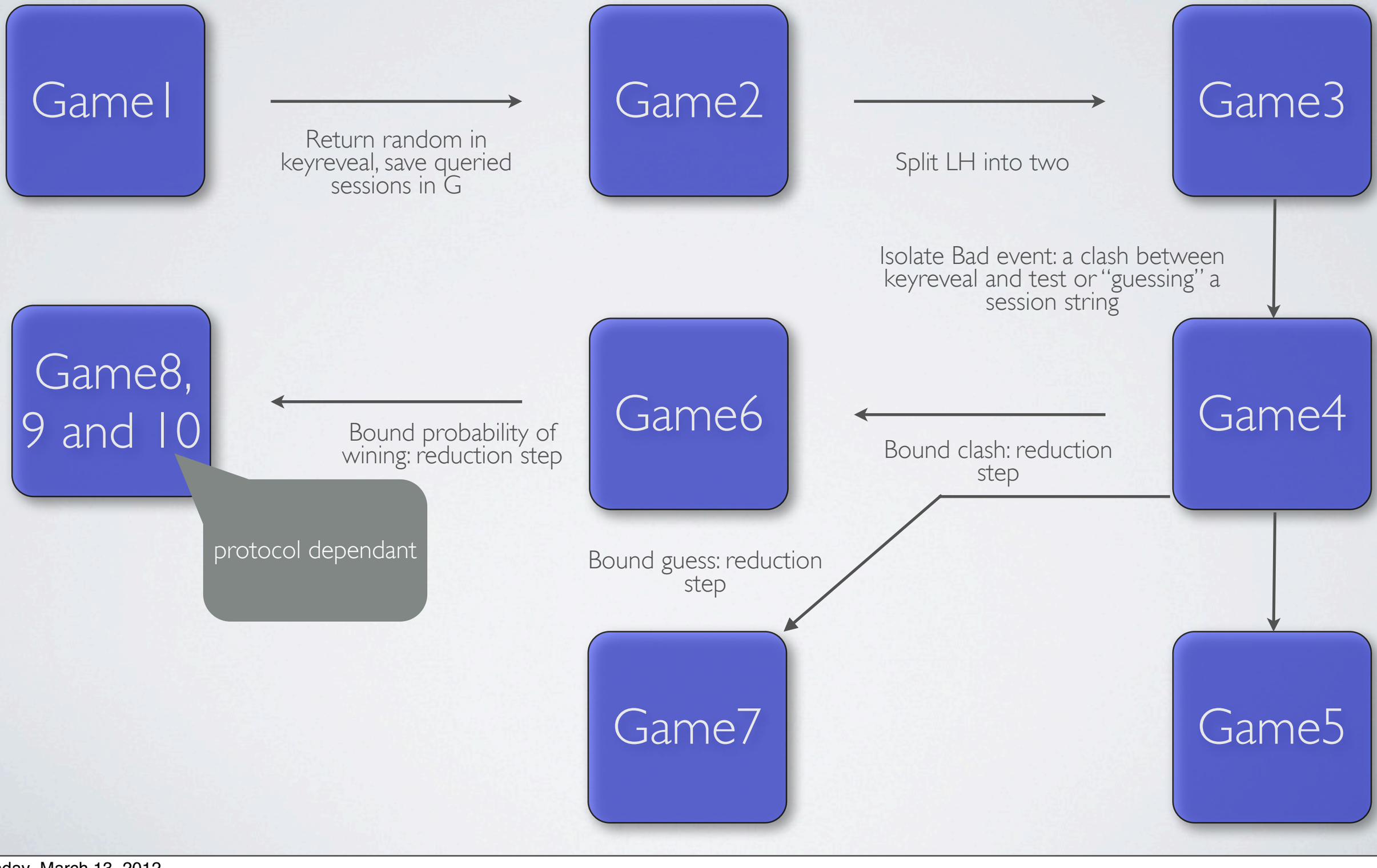
GAME SEQUENCE



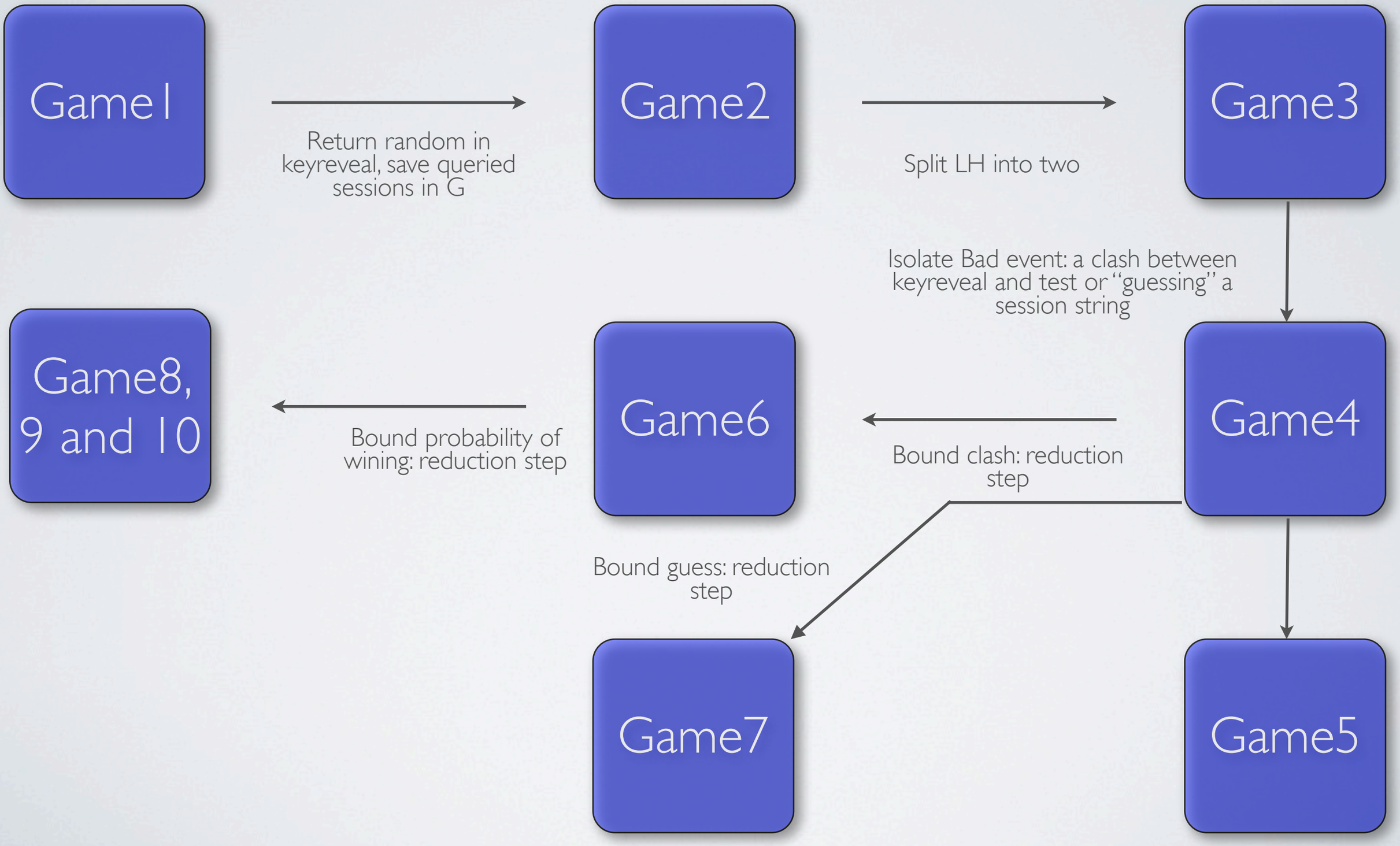
GAME SEQUENCE



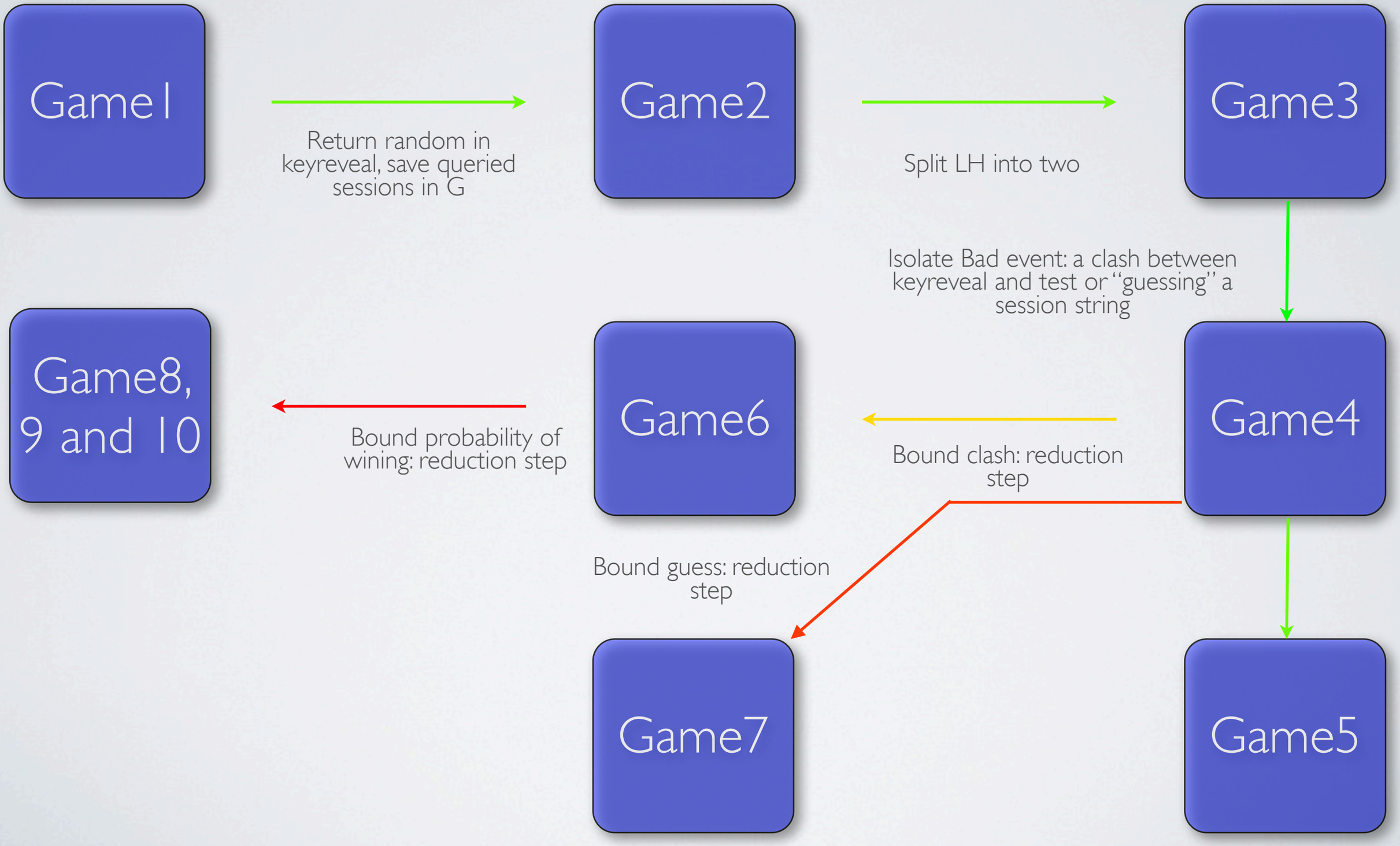
GAME SEQUENCE



GAME SEQUENCE



GAME SEQUENCE



AN INVARIANT (STEP 1-2)

```
invariant1(LH{1},LH{2}) && invariant2(G{2}, LH{1}, skey{2}, seed
{2}) && invariant3(G{2},LH{1}, LH{2}, skey{2}, seed{2}) && ={b,
bad, complete_sessions, incomplete_sessions, corrupt, pkey, skey,
seed,  LHT, tested_session,s}
```

```
pred invariant1(...)= forall (str : session_string), in_dom
(str,LH2)  =>  (in_dom(str,LH1) && LH1[str] = LH2[str]

pred invariant2 (...) = forall (str : session_string, fer_sid :
session_id),(( findelse_g_abs(G2,str,skey2, seed2) <> None) =>
( in_dom(str,LH1) &&  LH1[str] = G2[proj(findelse_g_abs
(G2,str,skey2, seed2))]))

pred invariant3 (...) = forall (str : session_string), in_dom
(str,LH1) => ( in_dom(str, LH2)  || findelse_g_abs(G2,str,skey2,
seed2) <> None)
```

CHALLENGES

- Scaling: many proof obligations, too much branching.
- Deal with Successive reduction steps in a more succinct way.
- Identify common proof steps and prove them once and for all.
- Invariant inference.